An optimised and generalised node for fat tree classes

By
Adamantini Peratikou

This thesis is submitted in partial fulfilment of the requirements for the award of the degree of **Doctor of Philosophy** of the University of Portsmouth

School of Computing
University of Portsmouth
Lion Terrace, Portsmouth, Hampshire
PO1 3HE, United Kingdom

Date: April 2014

DECLARATION BY CANDIDATE

I hereby declare that whilst registered as a candidate for the above degree, I have not been registered for any other research award. The results and conclusions embodied in this thesis are the work of the named candidate and have not been submitted for any other academic award.

Signature: *A.Peratikou*

Date: 04/04/14

Word count: 31059 (Without appendices)

Total word count: 37972

Abstract

Fat tree topologies have been extensively used as interconnection networks for high performance parallel systems, with their most recent variants able to easily extend and scale to accommodate different system sizes and requirements. While each progressive and evolved fat-tree topology includes some extra advancements compared to the original ones, these topologies do not fully address or resolve the issues that large scales systems face.

We propose an extended Zoned node, architecture as an alternative to conventional fat trees, and other variants. The extension relates to the provision of extra links to balance the number of routing switches, and hence increases the bisection bandwidth, and also to the extra layers that provide an inherent fault tolerance. In this work we emphasize on controlled power consumption, managed network complexity, faster message transmission, lower latency and higher throughput. These features are desirable for high performance parallel systems. We will show through semantics that our zoned node specifies most variants of the fat trees topologies such as k-ary n-tree and XGFT. We will also show that a replication of several zoned nodes into super nodes takes a broader view of complex interconnections such as dragonfly and PERCS.

We propose a generic source routing algorithm that we call in this thesis “the single-sliced-addressing” that works for all the classes of the zoned nodes, and we will prove through analysis and simulation that it outperforms the previous routing scheme deployed in some variant fat tree topologies. Most variants of fat-tree topologies do not address optimisation issues. In this work, we develop an optimising system that identifies parameters and components of the zoned node that lead to an optimized architecture. The optimization process is achieved based on minimising the overall cost of the network that is directly related to its complexity and therefore proportional to the relative power, which serves as the objective function that is minimised based on the traffic constraints to maintain a lower delay and a higher throughput. The simulation results show that the extracted optimised zone node performs well under various load conditions and traffic patterns compared to non-optimised variants of fat tree topologies.

I dedicate this thesis to

my parents Yiannos and Skevi ,

for their unconditional love and for

guiding me to where I am today.

Acknowledgements

There are a number of people without whom this thesis might not have been written, and to whom I am greatly indebted. I apologize that I cannot acknowledge everyone by name here.

Special thanks to my supervisor Dr. Mo Adda for the enormous help he offered me in order to overcome all the difficulties, which I faced during the accomplishment of this research. I admire his vast knowledge and enthusiasm in parallel processing field. Dr. Mo is the funniest advisor and one of the smartest people I know. I hope that I could be as lively, enthusiastic, and energetic as him. I would also like to thank him for his enormous patience, as I am a hard to handle stubborn individual.

I am very grateful to my parents Yiannos and Skevi for their constant support and for and for teaching me to aim high. Without their love, encouragement and generous financial assistance i would not have been able to finish this work. I cannot find enough words to thank them and whatever i say will not be enough.

I would also like to thank my boyfriend Marinos who helped me all these years in his own way, and for constantly cheering me up, my sister Mary who kept bugging me with her own problems and concerns, making me forget my stress.

I am also thankful to Thanos for listening to me constantly complaining, and for the various insightful discussions we had. I would also like to thank my fellow colleagues for their advice and encouragement.

I also appreciate all my friends and family members who stood by my side all these years.

Table of Contents

List of Tables

List of figures

Abbreviations and Symbols

ASIC	Application-specific integrated circuit
CPU	Central processing unit
CX- HCA	Core direct host channel adapters
DST	Destination
FFT	Fast Fourier transform
GFT	Generalised fat tree
GPU	Graphical processing unit
HTN	Hierarchical torus network
IBA	Infiniband architecture
MIN	Multistage interconnection network
NOC	Network on Chip
SF	Store and Forward
SIL	Side identity label of the switch
Snode	Super node
SRC	Source
UMIN	Unidirectional multistage networks
VIA	Virtual interface architecture
VTC	Virtual cut through
WDM	Wavelength division multiplexing
WH	Wormhole switching
XGFT	Extended generalized fat tree
Znode	Zoned node

Symbols that appear in equations and graphs

$\circ$	Recursively, replication.
$\stackrel{\text{def}}{=}$	Definition or representation. (Not a mathematical formula)
m	The number of Znodes in Snode
l	Layer
R_i	Routing Switch at level i

z_i	Number of Zones under level i
C_i	Connectivity function between level i and level i+1 by default $C_1 = 0$
ψ_i	Connectivity degree between level i and level i+1 by default $\psi_i = 0$
ψ^+	Positive forward connectivity degree
ψ^-	Negative backward connectivity degree
$oZ_n^{(l)}$	Optimal Znode of n levels with (l) layer
G_i	Number of groups at level i
L_i^+	Total number of links in positive connectivity
L_i^-	Total number of links in negative connectivity
L_i^d	Downward links for level i
L_i^{d+}	Down links at level i (positive connectivity)
L_i^{d-}	Down links at level i (negative connectivity).
L_i^u	Upper links at level i
L_i^{u+}	Upper links at level i (positive connectivity).
L_i^{u-}	Upper links at level i (negative connectivity).
T_i	Latency from level 1 to level i
Δ	Propagation delay
ρ	Routing bit
h_i	Number of Side channels at level I, to connect Znodes inside the Snode
$G_i^{(z)}$	Connectivity function applied inside Znode
$G_i^{(s)}$	Connectivity function applied inside Snode

Disseminations

This section of the thesis focuses in listing the publications, abstracts, presentations and posters that resulted from this work.

1. Peratikou, A. & Adda, M. (2014). Optimising extended generalised fat tree topologies. *Distributed Computer And Communication Networks 17th International Conference*, Oct 2013 Moscow, published in proceedings.

2. Peratikou, A. & Adda, M. (2014). Optimisation of Extended Generalised Fat Tree Topologies. *Springer*. Published in Communications in Computer and Information Science Volume 279, 2014

3. Poster presented at the University of Portsmouth Student conference on March 2012.

4. Presentation in University of Portsmouth School of Computing research seminar, on May 2012

5. Presentation in Frederick University Cyprus, on April 2013.

6. Poster presented at the University of Portsmouth Student conference on March 2014.

CHAPTER 1. INTRODUCTION

1.1. Background and motivation

To name but a few, studies and advances in fields such as astrophysics, evolutionary biology, chemical separations and atmospheric sciences have put more demand on the computation power of supercomputers. Current supercomputers have scored performance in the scale of petaflops for instance China Milk way achieve with an excess of 11000 processors a peak performance of 27.7 Petaflops (Xie et al, 2012), Cray XT7 has a peak of 2.3 Petaflops (Cray Inc, 2011), and Cray XK7 has a peak of 17.59 Petaflops (Cray inc, 2013). IBM research lab designed a third series of BlueGene machines, called BlueGene/Q which can scale up to 20 petaflops (IBM, 2014) superseding the Cray XK7. The next generation of supercomputers currently named as Exascale computers will adventure in tackling far more complex and larger programming problems, and open new horizons in the scientific, engineering and business disciplines. Such marvelous expectations will put more burden on the network interconnections to satisfy a follow up communication performance and power consumption. Indeed an increase in the computational speed of processors can lead to extreme power constrains (Bekas, 2013). Furthermore, exascale is expected to minimise the memory per core and per flop which will increase the need for latency control algorithms as processors will be thousands of cycles away increasing the waiting time to complete operations thus putting a limit on the overall performance (Gropp, 2013). Interconnecting a larger number of chips while keeping the overall cost under budget, and delivering the required performance are a trade-off. Thus a need to interconnect those high speed processing elements in the best manner possible is a challenging task. While the technology for higher speed processors increases, the research on the interconnection element of the high performance computing stays constant without any significant contributions to focus in overcoming the bottleneck caused.

The Fat tree topologies with its simplistic properties and ability to scale make it the most popular solution towards petaflops and exascale processors. Current trends Milkyway/ Thianhe and Infiniband utilise fat tree architectures. The high path diversity in fat trees

allows a fault tolerant and redundant system, and especially in high speed processors where the path diversity can play a critical role in the performance of the system due to the extremely high input rate. A number of variants of fat tree topologies were introduced over the years, bringing a hard dilemma in the parallel processing community, which fat tree configuration can offer high performance without enforcing a high power consumption. The question still remain unresolved as an attempt in optimising all the possibilities of fat tree and find the optimum one is required. We address these issues in this thesis.

Fat tree family architectures include some interesting characteristics such as scalable bisectional bandwidth and reaching maximum performance with a simple routing algorithm. They do experience however, some known issues such as the bottleneck caused from the limited availability of paths as in some cases only a single path exists, especially in the lowest leaves of the tree. Moreover the cascaded hierarchical nature of fat tree topologies can cause a dramatic increase in the network complexity; and hence on large systems, the overall cost of the network and the power consumption escalates. Another constrain of fat tree class architectures is caused by the routing schemes such source-destination addressing, Infiniband (Bogdanski 2013) and destination-based (Gomez, 2008). The latency that those routing algorithms becomes very apparent when the size of the network upsurges. The endeavor of the routing elements to find the paths will rely on the messages carrying larger overheads, or checking the lookup tables to identify the directions of the flows bares cumulative higher latencies.

In this research, we propose a generalisation of fat tree topologies that aims to experiment with different configurations to achieve the desired requirements. Our aim to achieve an efficient and flexible topology is supported by a simple source routing algorithm to minimise the latency in the network, and by replication process of the basic topology several times we obtained better performance for larger systems and resolve the scalability issues of some fat tree topologies such as k-ary n-tree networks, while improving bisectional bandwidth.

The multiplicity concept that we adopt in this research allow us to replicate basic structures to form an even higher scalable node called the super node. Such node, like

dragonfly (Kim & Dally et al., 2008) and PERCS (Arimili et al., 2010) can be deployed to interconnected large scale systems, with even larger number of processor without major upheaval to the performance of the system. The major aim of this thesis is the development of an optimised zoned node architecture that can provide high performance computing base and then replicate it to scale to accommodate larger number of processing power, and to show how to generalise and optimise fat tree classes under the configuration of the zoned node architecture, using several design parameters.

1.2. Approach, original contribution and benefits of this work

The common trend in architecture research is to improve network performance by minimizing the latency, minimize power consumption, and maximise the throughput. These factor are inter-related. Keeping the latency bounded usually increases tremendously the overall physical implementation complexity cost and therefore mounting the power consumption. Power consumption must be considered as a technique that can better explore the utilization of Routing switching in multi-core architecture and generate performance improvement for future core systems. On the other hand, due to the nature of the queuing networks, an excess in throughput is always followed by an increase in network latency. It is important to study what levels of latency can be tolerated by a set of applications, and to investigate how the interconnections should be designed to guarantee a certain level of performance by minimizing power dissipation, within the offered load. Moreover, with the increase in the number of levels of cores an intelligent design of a multiple levels switched network becomes an attractive factor.

We present zone node network architecture and discuss its short-term challenges and long-term implications, on the current and future research trends.

The principal contributions of this thesis involve the following:

- Proposal of a zoned node architecture with alias “Znode” that generalizes fat tree classes and can be used as a building block for large scale parallel systems. Under the Znode architecture we introduce an additional parameter that refers to the number of layers, in order to increase the capacity of the system and evenly distribute the traffic. We also introduce a connectivity degree along with a simple

source routing algorithm that uses a single bit direction with sliced port labels. The sliced routing algorithm is generic and can be adopted in ay fat-tree based environment.

- Proposal and development of an optimisation model to optimise fat tree classes, including the Znode. To the best of our knowledge this has not been considered for current fat tree variants. Our optimisation model includes a set of constrains that can quantify all performance parameters and provide the best configuration for our fat tree based topologies. This optimisation model and its associated parameters make up as a tool for performance evaluation and analysis that minimises the power consumption and maximise the overall performance.

1.3. Objectives and thesis outline

The main objectives of this thesis are to:

- Come up with a new generation of fat tree class topologies to cope with the current and future performance requirements, the ground for high performance supercomputers.
- Propose a new model based on fat tree classes with low latency, low relative power consumption, and higher throughput.
- Provide an optimising mechanism to select components and structures of the networks that exhibits the lowest cost possible while maintaining higher performance.

The main chapters in this thesis are organised as follow:

- Chapter 2 discusses the background and the related work in the context of our proposed interconnection architecture. It addresses the main issues in the area such as the high power consumption, low scalability along with the communication latencies and overheads of current architectures. Chapter 2 shows the necessity towards a generalised approach to fat tree topologies, to fully address and resolve the issues that faces large scales systems. Moreover, an overview of the previous work in the area of high performance architectures is given, along with the challenges of each architecture.

- Chapter 3 gives an overview of the Znode architecture, by analysing its topological advantages along with its salient features that increase the performance without increasing the power utilisation. A definition of the architecture is provided along with a description its replication process and connectivity.

- Chapter 4 illustrates a new source routing algorithm scheme called “single bit sliced addressing” and explains the addressing utilized. The optimization of Znode and other fat tree variants is also provided as a tool to increase the performance even further.

- Chapter 5 presents a simulation tool that can be used to evaluate the effectiveness of Znode architecture. The importance of this chapter is the performance gained under Znode architecture compared to a set of widely used architectures such as K-ary n-tree. Some numerical and simulation results have been presented for different scheduling, distribution and processors number scenarios that demonstrate the applicability of Znode architecture on large scale systems.

- Chapter 6 explains the technical issues along with the hardware feasibility for Znode architecture. One of the main goals of this chapter was to detect the appropriate hardware components that can fully implement the proposed architecture realistically.

- Chapter 7 concludes the work in the area of high performance interconnection networks, by evaluating the findings along with providing possible improvement suggestions for future work and further research.

CHAPTER 2. LITERATURE REVIEW

In this thesis, it is stated that a good plan with the use of multiple levels of cascaded SWITCHED trees can be used to minimize power consumption of long interconnects with the use of “side links” to achieve higher performance on large traffic. This work illustrates a hierarchical Cascaded network of switches that suits the particular requirements thus Portsmouth parallel machine which is named Zone tree architecture is proposed. The main advantage of this is its low latency specially when considering high traffic locality.

This chapter analyses and reviews the current trends in parallel architectures.

2.1. Background information and terminology

This chapter specifies the different definitions used throughout this thesis to serve as a background information dictionary.

2.1.1. Parallel processing

Parallel processing is the simultaneous use of multiple compute resources to solve a computational problem. In parallel processing a problem or a task is divided into multiple pieces, these pieces are the instructions or threads depending on the parallelism, which will be processed by the Compute processing units. Those pieces of the problems can be solved simultaneously in a lot less time than a single processing computer. Parallel processing is used in order to save time and money as large problems will take an enormous amount of time to be solved in a single processing unit. Parallel processing solves them simultaneously (Barney et al., 2010).

To achieve high performance, contemporary computer systems rely on several forms of parallelism such as data parallelism, instruction-level parallelism and thread-level parallelism. Wide-issue superscalar processors exploit by executing multiple instructions from a single program in a single cycle. Multiprocessors exploit by executing different threads in parallel on different processors.

2.1.1.1. Instruction level parallelism

A problem is broken into a stream of instructions or several threads of execution. Instructions are executed one after the other. Processors maximize the execution of the number of instructions per second, by using different design configurations.

2.1.1.2. Thread level parallelism

Thread level parallelism or simultaneous multi-threading is when multiple threads are executed at the same time without the need to be broken down into instructions. It is basically the utilization of functional units further by sharing functional units between more than one thread.

2.1.1.3. Processor types

Pipeline and Superscalar processors try to overlap several instructions and complete more instructions per unit of time. Multicore is the multiple separate cores that are connected together by an on chip network. Similarly a multiprocessor system consists of multiple processing units connected together via an off chip interconnection network.

2.1.2. Parallel architectures

2.1.2.1. Shared memory

Shared memory architecture typically accomplishes interprocessor coordination through a global memory shared by all the processors. The main problems of shared memory architecture are the low scalability and the high cost. Uniform Memory Access (UMA): Equal access to the memory, using symmetric multiprocessors (SMP), with cache coherence CC-UMA. Non-Uniform Memory Access (NUMA): One SMP can access shared memories, through a communication network, giving unequal access time.

2.1.2.2. Distributed memory

Distributed memory architecture typically combines local memory and processor at each node of the interconnection network. Message passing is used to communicate between any two processors, and there is no global memory.

The main constrains of Distributed memory architecture is that, the memory access is required to be explicit, which is embedded with the use of message passing interface (MPI). It is a cost effective – use off-the-shelf processors, it has independent access to the memories, no cache coherence needed. Although distributed memory can provide scalability in large proportions it is difficult to map all data structures of problems into it. Additional efforts are required by programmers to map algorithms into the configuration, which can sometime cause more overheads.

2.1.2.3. Hybrid-distributed shared memory

Hybrid distributed shared memory employs both shared and distributed models. Usually the shared part uses cache coherence. Set of SMPs sharing a global space among themselves and communicating via the message passing with others.

Our proposed Znode architecture is a platform for hybrid distributed memory where cores relay on SMP sharing space. The main advantages of hybrid distributed shared memory is the increased scalability that it provides, compared to the previous architectures.

2.1.2.4. Interconnection networks for parallelism

- **Switch based** interconnection networks are networks were the connections among processors and memory modules are made using simple switches, three basic interconnection technologies exist crossbar single-stage and multi-stage.
- A **crossbar** network can provide simultaneous connections among all its inputs and all its outputs. The crossbar contains a switching element at the interconnection of any two lines extended inside the switch. The message transfer delay in a crossbar network is constant regardless of which input and output are communicating.
- **Single stage** networks consisting of only one stage of switches between the inputs and outputs of the network.
- **Multistage** interconnection networks consist of a number stages each consisting of a set of switching elements, which are connected together in stages.

Zone node architecture is implemented with a multistage interconnection network. Higher and lower level networks are connected together via simple switching elements, in a cascaded tree approach. The root or the top of the cascaded tree consists the highest level network, while as we move down the branches the lower level networks exist. The

processing units are attached in the bottom or base of that tree. This type of tree are known as fat tree interconnections.

2.1.3. Network characteristics

Network characteristics are parameters that can affect the overall performance of an interconnection network in terms of cost, message delay and power consumption.

2.1.3.1. Network diameter

Network diameter is the larger minimum distance between any nodes. It is measured by hops between any two nodes in the network. It can be used as an approximate upper bound of the packet latency. The smaller the diameter the better is the packet delay.

2.1.3.2. Node degree

Node degree can be defined as the sum of the in and out degree of the node. That sum can affect, and determine the scalability of an interconnection network. Networks can have either a fixed constant degree or a non-constant degree that can change according to the configuration of the nodes on the system. A popular interconnection architecture with a non-constant node degree is hypercube topology. Constant node degree, is when the node degree does not change when the nodes increase or decrease, such as torus topology.

2.1.3.3. Latency.

Latency is a part of the actual delay or the time that is needed for a packet to get from node A to node B in an empty network. The latency can be affected by the switching elements, the software overheads, the routing delay and the connection delay. Sources of latency can include propagation delay and time that the network card needs to place the pulse on the wire and interpret it.

2.1.3.4. Queuing delay.

Queuing delay is the time the packet waits at an output link for transmission. It depends on several factors such as the congestion level of the router. If the traffic keeps arriving faster than it can be processed then inevitably the buffers will fill up and packets will be dropped, causing data loss.

2.1.3.5. Processing delay.

Processing delay is the time the routers or switches take to process the packet header information. Sometime referred to as routing delay.

2.1.3.6. Transmission delay.

Transmission delay is the time that is required to push packet bits into the link /channel, it is related to the bit rate of the network card.

2.1.3.7. Network congestion.

Network congestion is the lowering of the quality of service caused when a link or node carries a lot of data.

2.1.3.8. Throughput.

Throughput is the number of packets successfully transmitted per unit of time, such as bits transmitted per nanosecond. It can be affected by factors such as network congestion from possible heavy usage and low bandwidth allocation between the network devices. The higher the throughput, the faster is the packet transmission.

2.1.3.9. Bisection width.

The minimum bandwidth, over all partitions of the network to its two closest parts or the minimum number of connections that need to be removed to partition the network into two equals and disjoint parts. The larger the bisectional width is, the thicker is the channel width, and therefore the lower is the latency. As the maximum channel width is determined by the division of the bisection width.

2.1.3.10. Path diversity.

Path diversity is an architecture advantage feature that can be obtained when more than one shortest path exist between the node pairs of that network. For instance butterfly interconnection networks don’t have path diversity while torus can exploit path diversity. Path diversity is useful to reduce network congestion and to increase the fault tolerance of the interconnection network along with better load balancing of the network.

Zoned node architecture provides path diversity, as more than one links are used to connect the switching elements. In the lowest level of the zoned node, processors are divided into multiple groups connected together by switching elements that provide

multiple paths to enter the network. Moreover, side links are also provided as a configuration option, to further increase the path diversity in a single super node configuration.

2.1.3.11. Cost.

Cost plays an important role in the selection of the interconnection network to be used in the parallel processing architecture. The number of links in the interconnection network increase the complexity of the network. Since the complexity of the network can increase the overall cost of the system with lower complexity, there is also lower cost.

2.1.4. Networks

2.1.4.1. Direct network

Direct networks can be defined as the networks where all routing switches are considered as end points. A direct connection between the processing elements exists as all the nodes have processors attached. Popular direct networks include k-ary n cube, n-dimensional mesh, torus and hypercube (Dally & Towles, 2004).

2.1.4.2. Indirect network

Not all switches are considered as end point such as butterfly networks. Widely used, indirect networks include butterfly, fat tree, k-ary n-tree, extended generalized fat tree and hyper tree. Our proposed architecture Zoned node is consider an indirect network as not all switches are directly connected to a processor.

2.1.5. Switching forwarding methods

A switching method describes how packets are moved from an input link, through a switch, and to an output link, how packets are buffered and what happens if the necessary resources are busy.

2.1.5.1. Store-and-forward

When a message is received the switch saves the frame of the packet into its on-board buffers and then computes the CRC-cyclic redundancy check. If the frame has errors or if it is larger or smaller, than the predefined threshold, then the frame is discarded. If the frame does not have any anomalies then the switch checks the look-up table to find the

destination interface specified on the header, the path is decided from the lookup table and the frame is then forwarded to the output link. In high speed networks, the CRC phase is usually not considered, and hardware routing is used for routing decisions.

2.1.5.2. Virtual Cut-through.

In this method the switching element copies the destination address of the frame first, it then looks on the switch look-up table to find the appropriate destination interface, and it forwards the frame based on the information specified on the lookup table. Similarly to store and forward, in virtual cut through buffers exist for the whole packet but instead of waiting for the whole packet to be buffered the header flit is cut through into the next destination once the path decision is made. Virtual Cut through method is generally preferred as it reduces latency, but on the other hand its main constrain it's that it decreases the bandwidth.

Zoned node interconnection network utilises virtual cut through routing as it is considered one of the best and most preferable switching technique. Virtual cut through is established in zoned node with the use of packet flits, which include an encoded address, of a single bit, as it will be shown later in the proceeding chapters.

2.1.5.3. Wormhole flow control routing.

A special case of cut through switching that removes the dependency between maximum packet length and buffer size found in store and forward and cut-through switching. When a packet arrives its destination address is checked immediately and the header of the packet is forwarded to the output port straight away if the output port is not used. There is no packet buffering thus the buffering memory is minimal, it can support rapid switching and packets of arbitrary size. The main constrain in wormhole routing is that if the output port is busy or unavailable the packet has to wait until it's available which causes blockage of all other packets that are waiting to use the links that this packet is occupying while waiting. An extended version of wormhole flow control includes virtual channels flow control as well, in which a number of channels are allocated in each input port to increase throughput an avoid deadlock. Due to the fact that in wormhole switching the packet is not required to be buffered it is far more efficient than other routing techniques, our proposed architecture also utilises the wormhole flow control routing.

2.1.6. Routing techniques

Any of the following techniques can be subject to variation to include the best features possible. For instance while oblivious routing does not include load balancing, with a combination of algorithms oblivious routing can then support load balancing.

The routing algorithms can be classified based on their adaptability to the different network states, their number of paths and their routing decision mechanisms. For instance adaptive routing algorithms can adapt on the network faults or network stages while deterministic algorithms cannot. Figure 2.1 provides a classification of the routing algorithms, based on their routing decisions whether source routing or distributed routing followed by the implementation of the routing decisions based on table lookup or arithmetic addressing.

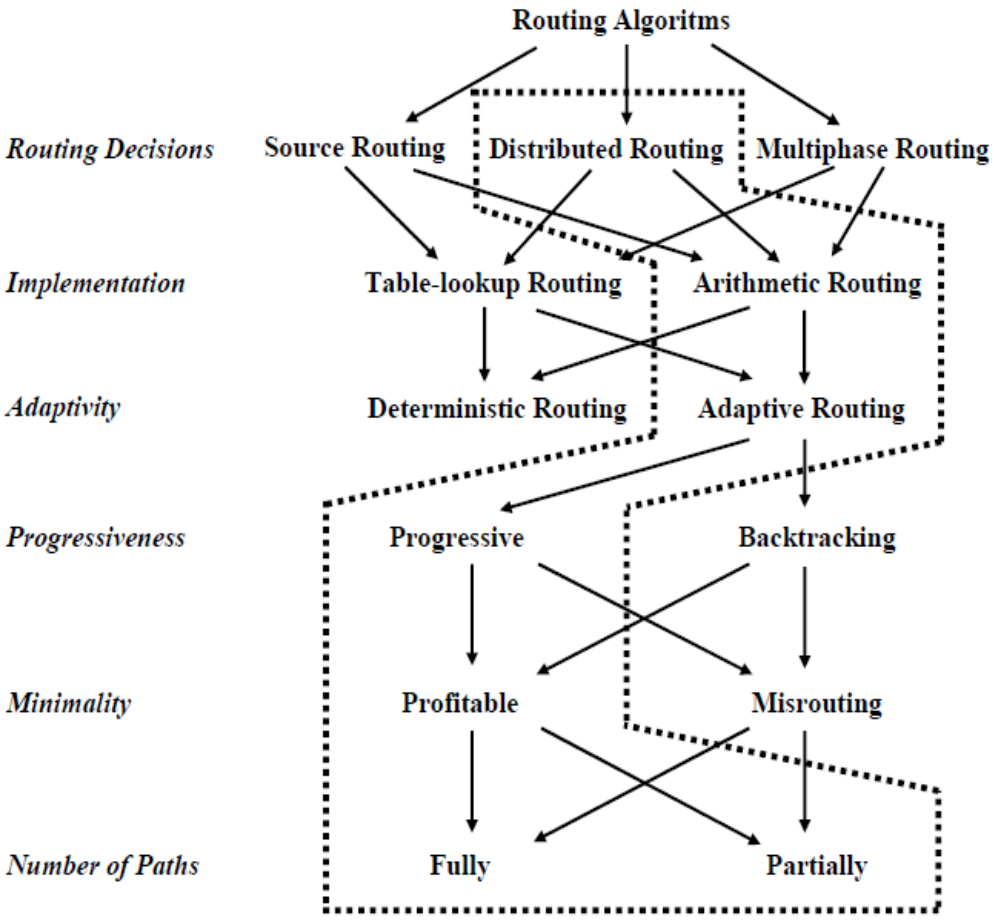


Figure 2.1. Classification of routing algorithms (Duato, 2003)

2.1.6.1. Oblivious routing.

In oblivious routing the path is defined in a unique manner by the source and the destination node. Network congestion is ignored when a path is being chosen using oblivious routing. Oblivious routing can keep the routing algorithms simple but it is unable to adapt. Global information is required in order for oblivious algorithms to calculate the best path thus can cause a lot of unwanted delays (Dally& Towles, 2004).

2.1.6.2. Deterministic Routing.

Deterministic algorithms are a subset of Oblivious routing as all packets from the source to the destination will travel the exact same path without considering the current state of the network. While this algorithm is simple and deadlock free, it has poor load balancing and it also eliminates the path diversity of the interconnection network. Examples of deterministic routing can include **XY routing** and **dimension order routing** as the packet travels dimension by dimension. K-ary n cube networks such as meshes and tori topologies use dimension order deterministic routing algorithms (Kinsy, 2009).

2.1.6.3. Adaptive routing.

In adaptive routing the path is chosen by the network conditions during the time of traverse. The path can change while the packet is traveling to avoid possible network congestions. In adaptive routing depending on the algorithm used it can adapt faster to faulty or disconnected nodes in the network allowing less congestion than oblivious routing. But it can also cause delays in delivery due to the high possibilities of path selection and the time needed for the algorithms to select a "good" path. Adaptive routing is generally preferred in indirect networks such as fat tree or folded clos topologies (Kaur, 2012).

2.1.6.4. Unrestricted or fully adaptive routing.

Unrestricted is the same as adaptive routing but in this case any neighbor node can be used in the path. Therefore increasing the path possibilities even more, which can cause more delays (Dally & Towles, 2004).

Zoned node interconnection network adopts both deterministic and fully adaptive routing. When packets traverse in the downwards direction, deterministic routing is used. While

when packets travel in the upwards direction adaptive routing is utilised to achieve higher path diversity.

2.1.6.5. Minimal Routing.

In minimal routing each link takes the packet closer to its destination. The routing is done in two stages the first stage is to route to random node and the second to route to destination. For each packet minimal routing randomly selects a node inside the minimal quadrant to route the packet from its source to that node and then from that node to each destination. An example of an interconnection architecture that performs better under adaptive minimal routing is K-ary n-mesh topology (Pasricha & Dutt, 2008).

2.1.6.6. Non minimal routing.

Non minimal routing allows the path to lead away from the destination in order to avoid possible congestion as non-minimal routing is not restricted to take the shortest path. But it can cause livelock as the packet can travel forever without ever reaching its destination. It includes routing algorithms such as virtual channel based routing, search based algorithms and turn based routing (Pasricha & Dutt, 2008).

2.1.7. Communication patterns

2.1.7.1. One to one Communication or point to point communication.

The information is only exchanged between one or several pairs of nodes. The information cannot be duplicated. One to one communication pattern can be defined as single pair communication that has no deadlock or network congestion as there is only one communicating pair with a shortest path routing algorithm. In many-one-to-one communications, a number of communicating pairs can simultaneously exchange information. In permutation routing, a node can be the source for a packet, and the destination for another packet at the same time.

2.1.7.2. One to many communication.

One to many communication pattern is known as multicast and can be defined as when a node can be the sender while all or several nodes are receivers at the same time. The

information can be duplicated between the nodes. When the information sent in each node is different, the pattern used is called one to all scatter.

2.1.7.3. All to all communication.

All to all communication can be named scatter or all to all broadcast. Similarly to one to many communication patterns they fall under the category **collective communication**. In this method a sender can communicate and transmit packets to all the nodes in a group in the network.

2.2. Review of related work and popular and parallel architectures

In this section, relevant prior work on multi core architectures will be reviewed. As the number of processing nodes to be interconnected increases, simple topologies like rings or meshes become inadequate. The increment in the number of nodes in these topologies considerably reduce their performance. To address the scalability problems of such topologies, several new network architectures have been proposed over the years.

2.2.1. Direct networks

Hypercube based interconnection network for Double loop hypercube was proposed by Youyao (Youyao, 2008). A three dimensional optical network that regardless of increase in the actual size can still maintain constant node degree. It uses the shortest path routing algorithm with the source node requiring the dimension size plus one rounds to transform a message through the network .On receive each node decides if it should be delivered to the local node or to be forwarded to an adjustment node. State information is not required for routing decisions therefore lower delays (Youyao et al., 2008). Extended folded cube architecture was proposed on 2011 by Mu et al (Mu & Li, 2011). It involves a new interconnection network that is a variation of hypercube by combining folded cube and extended hypercube architectures, forming a hierarchical recursive nature interconnection network. Folded cube (El-Amawy & Latifi, 1991) and extended hypercube (Kumar & Patnaik, 1992) are two architectures that variate the hypercube technology in an attempt to obtain a constant node degree, and achieve a more scalable system. Extended folded cube consists of two levels of hierarchy and can be constructed by attaching extra links to the hypercube, (similarly to folded cube) which will divide by half the network

diameter and achieve lower delay in communication links along with lower overall cost compare to standard hypercube. The differences between folded cube and extended folded cube is the hierarchical levels along with the fact that the processing elements in extended hypercube are able to concentrate on the computation as more network controllers are added that are responsible for the communication (Mu & Li, 2011).

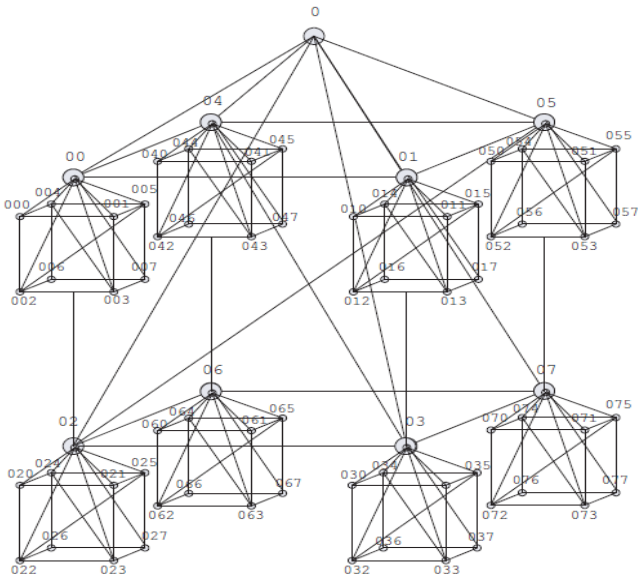


Figure 2.2. Extended folded cube (Mu, 2011)

Extended hypercube keeps the good features of hypercube but lowers the network diameter along with introducing constant node degree. On the other hand extended folded hyper cube uses the same topological features as extended hypercube while combining folded cube extra features in a hierarchical tree (Mu & Li, 2011).

As illustrated in figure 2.2 extended folded cube consists of a number of channels per node, but the paths that a packet will have to be routed from one dimension of the inner cube, to another dimension of the outer cube, will be unnecessary complicated, thus the routing techniques for that topology will be too time consuming with a high chance of multiple live locks. Furthermore, as the processors increase the cost of adding a controller per module will be unbearable on the system, and also the fault tolerant properties need to be revisited. The authors only provided performance analysis results of extended folded cube, compared to hypercube and folded cube, without comparing their proposed solution with other more popular architectures such as torus and k-ary n cube.

Symmetric Tori connected Torus Network (STTN) is a hierarchical interconnection network that consists of a number of hierarchically interconnected basic modules (BM) in order to construct a high level network (Al-Faisal & Rahman, 2009). Each basic module consists a 2D torus network with 2^{2m} processors. Where m can be equal to 1 or 2. "Each BM has 2m+2 free ports at the contours for higher level interconnection. All ports of the interior nodes are used for intra-BM connections. All free ports of the exterior codes, either one or two, are used for inter-BM connections to form higher level networks. BM refers to a Level-l network (Al-Faisal & Rahman, 2009)."

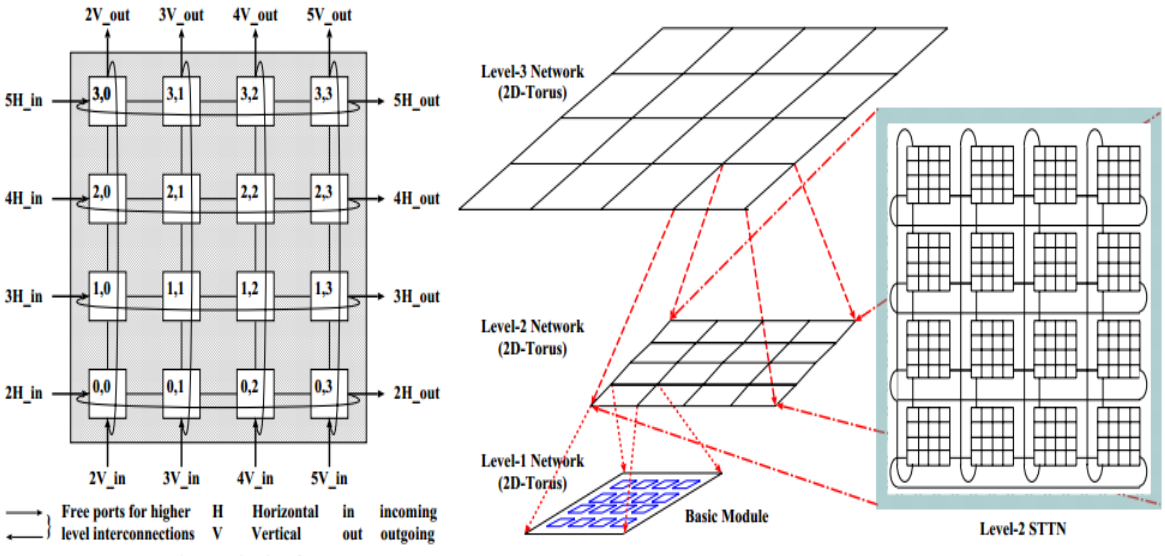


Figure 2.3. Interconnection on STTN and higher level network. (Al-Faisal, 2009)

Even higher networks in STTN can be constructed by recursively interconnecting lower level networks in a 2D torus as it can be illustrated in figure 2.3.

Each higher level interconnection network basic module (BM) uses "4 x (2^q) = $^{2q+2}$ of its free links, 2(2^q) free links for vertical interconnections and 2(2q) free links for horizontal interconnections. Here q ∈ {O, 1,....., m} is the inter-level connectivity (Faisal et al 2009)"

The nodes are addressed using base 4 numbers such as the first digit illustrates the row index and the second the column index. On larger Level-L STTN networks the addresses are represented as A = A ^{L} A $^{L-I}$ A $^{L-2}$……. A ^{2} A ^{1}. Therefore the total digits of the addresses in each higher level is n=2L. The node degree n STTN is 6 due to the fact that each BM has six links. According to the authors performance tests it is shown that STTN

has much lower average distance when compared to TESH TORUS and MESH interconnection networks and when simulating 1 million nodes the authors found out than STTN average distance is quite controllable and far better than other similar architectures.

STTN architecture is a good solution in high performance systems due to its scalable nature as the node degree is always 6, the system can expand or decrease as required. Compared to torus and meshes interconnection networks the cost of STTN is lower. But on the other hand the equations given by the authors for the total network diameter in their architecture include a lot of unwanted elements and the final network diameter will be large resulting in high message delay. Moreover the bisection width of STTN is far larger than other torus and mesh interconnection networks, which will imply that a lot of extra chip wires will be required for the physical implementation of STTN therefore higher cost and power consumption is inevitable.

Torus embedded hypercube interconnect proposed by Kini (Kini et al., 2009) is a network architecture that can offer scalability in parallel computation. The basic idea is the combination of torus and hypercube networks together to achieve a low diameter and a more scalable architecture. In parallel computation architectures, the interconnection network needs to be able to scale. Since hypercube offers a small network diameter, simple routing algorithms and high connectivity but its growing node degree make it impossible to build a scalable architecture. While torus has a constant node degree and can offer highly scalable architecture but its diameter is quite large. The authors decided to embed the two interconnects together to implement a small diameter low degree system thus a reduced hardware cost.

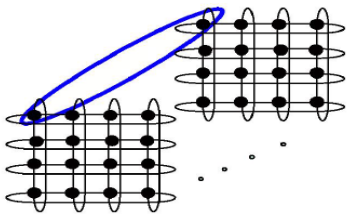


Figure 2.4. Concurrent torus to form a hypercube (Kini et al., 2009)

A set of concurrent nodes groups that form a torus, once combined together they can form a hypercube (shown in figure 2.4). The nodes of hypercube that have similar addresses will be the nodes of each individual torus illustrated in figure 2.5. According to the authors

(Kini, 2009) If *l x m* is the size of the torus, then N is the number of nodes connected in the hypercube. Therefore a (l,m,N) torus embedded hypercube network will have $l \times m \times n$ with each node address represented by (l,j,k). For instance 8 concurrent torus, with 2x2 size can form 4 hypercubes that consist of 8 individual nodes, thus the total nodes will be *2x2x8=32*.

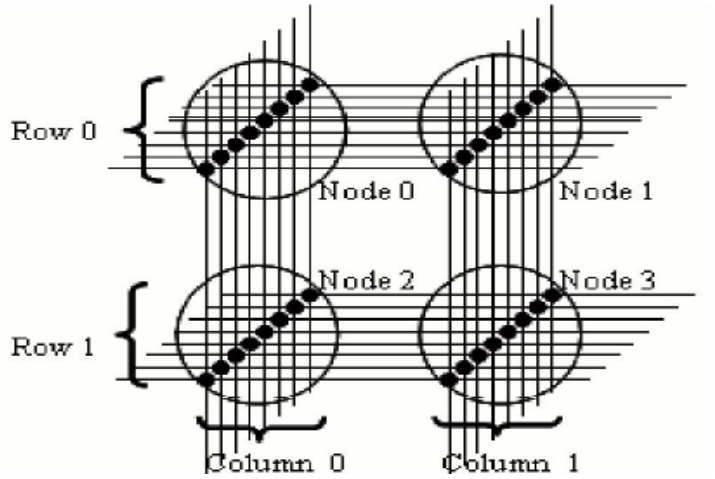


Figure 2.5. A 2x2x8 torus embedded hypercube network (Kini et al., 2009)

While the torus embedded hypercube approach can reach better scalability and decrease the network diameter, the results obtained according to the authors do not reach the prominent features for either one of the networks. Furthermore for this solution to work, the size of hypercube is required to stay constant at all times to avoid node degree changes, but with the emerging technology, the needs for a more scalable system that can be expanded accordingly do not favor this approach.

Rahman et al, (Rahman & Horiguchi, 2003), proposed a pruned hierarchical torus network to increase performance by reducing complexity. The hierarchical network (HTN) consists of 3D-tori modules connected together by a 2d torus network. Dimensional order routing algorithm is utilized in this network (Rahman & Horiguchi, 2003). The wrap-around links used in the 3D tori, cause high complexity, thus pruning the HTN to remove a number of physical links without complicating the routing algorithm was proposed by the same author later in literature (Rahman & Jiang et al., 2009). The purpose of pruning is to remove the links from a basis network in order to simplify it in terms of scalability and modularity and to overcome the limitation on bandwidth by decreasing the diameter as illustrated in figure 2.6.

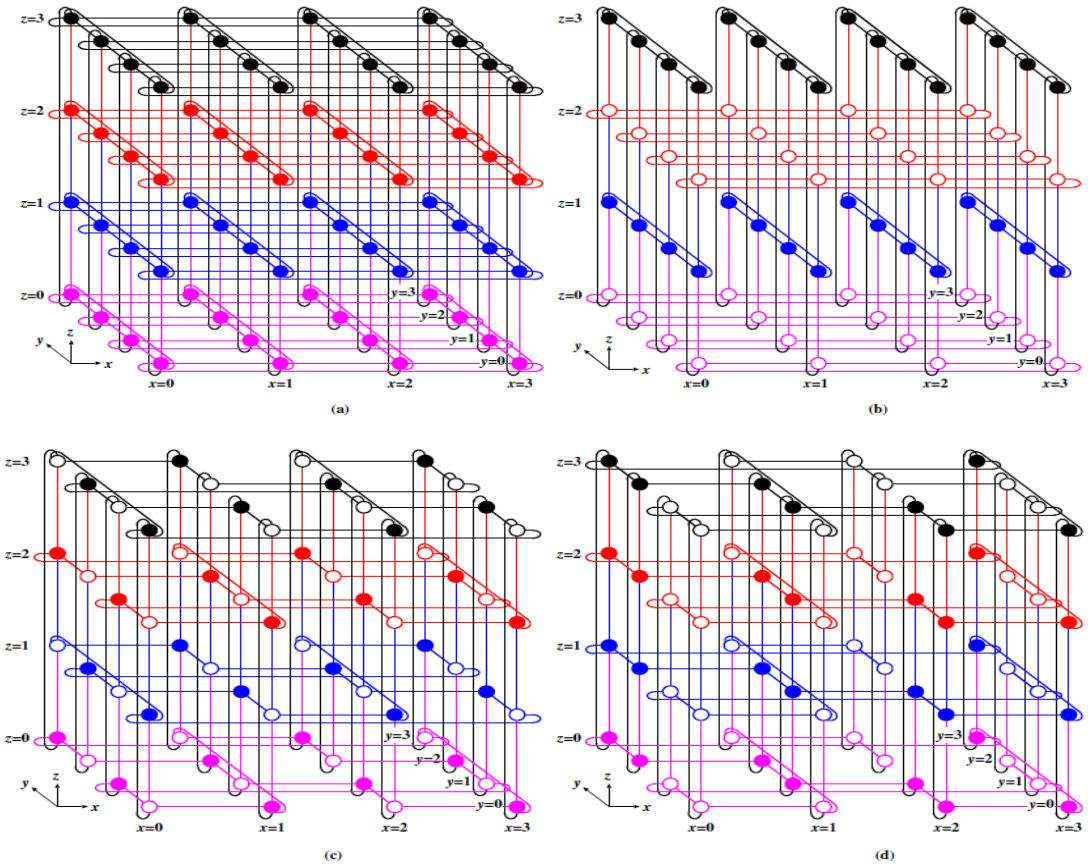


Figure 2.6. (a) Unpruned hierarchical torus. (b) The same torus network as figure 'a' but pruned in z dimension (c) pruned along the x, y and z direction (d) Pruned along the x and y direction forming a disjoint network (Rahman, 2009)

The Pruned HTN illustrated better results than HTN in terms of power consumption, latency and bandwidth. However there is not enough evidence on the results of high-traffic networks. Furthermore the pruned approach is a promising solution for all types of cascaded tree-like networks such as k-ary and torus. It can also be considered in the future work of our proposal zoned node, as an enhancement to the current configuration by reducing the links that are not required.

Gemini, (Alverson et al., 2010) is an interconnection network for the new Cray XE/XK system that consists of a direct 3D torus topology on 225,000 cores that claims a higher performance with lower latency. Multiple links are provided by torus between each node, so that packets are adaptively distributed in all the links to achieve higher performance. The total latency is determined by the end points latency's and the number of hops that the packet has to undertake in each cycle until it reaches its destination. Packets hold the destination information along with the details on how it will be routed. The address is uniquely specified using an 18bit slot in the header of the packet. Compared to BlueGene,

as both use torus their difference is that Gemini forms a single network with high bandwidth links while in BlueGene the bandwidth is divided into two networks, the one of them is torus and the other one is tree. Gemini uses virtual cut through for the outside links and wormhole for the inter-routing links. While it achieves a higher throughput, than other similar approaches it is mainly an internal design, as it uses novel routers and NIC that do not currently exist in hardware, while our solution consists of an interconnection approach to achieve higher performance.

Tofu or "torus fusion" is a six dimensional mesh/torus topology developed by Fujitsu to provide the scalability to generate a supercomputer with more than 80,000 nodes (Ajima, 2011). The six dimensional mesh/torus interconnect is formed from the Cartesian product of abc and xyz. As each network node consists four fixed size links for abc 3d mesh/torus and six links for xyz 3d torus, adding up to 6d. Each mesh torus is interconnected by 12 links. The routing of tofu involves three traverse stages. First it goes through abc-axes them to xyz-axes and finally through abc-axes again to ensure path diversity. Each routing header has two sets of abc coordinates (Ajima et al., 2012). Four virtual channels are used in Tofu interconnect, two virtual channels are used to separate the request and respond packets, one for the abc-axes traverse and one virtual channel is used for the prevention of deadlock on xyz route. Each virtual channel has 8 kilobytes of receive queue, the sender is notified for the space in the queue before packet transfer. The fact that each virtual channel has 8 kilobytes receive queue and the notification is written on the packet header can cause a dynamically increasing packet size due to all the multiple paths that the packet will have to traverse until it reach destination.

Hyper node torus is an architecture that extends torus interconnection by replacing each node in the torus with a ring. As it can be illustrated in figure2.7, Hyper node torus has two different levels of structures, a low level (z) that shows the node position in the ring and a high level (x,y) that shows the node position in the torus (Khosvari et al., 2011). Distributed architecture is used therefore allowing the network packers to be shared between the elements as each node can either be a memory or a processor. The total number of nodes is N=2nk^n. XY routing is used along with the packets being delivered from source to destination based on shortest path algorithms along with wormhole routing to help in hiding the message passing delay from the network. Adjustments to the XY routing were also proposed by the authors (Khosvari et al., 2011) to achieve the optimum routing for hyper node architecture. "In each step that packet will be routed, before

comparing the (x,y) coordinates of current node to the (x,y) coordinates of destination node, first z coordinates of both nodes should be compared, then after selecting out path from the ring, routing will be based on comparing (x,y) coordinates of source and destination node" (Khosvari et al., 2011).

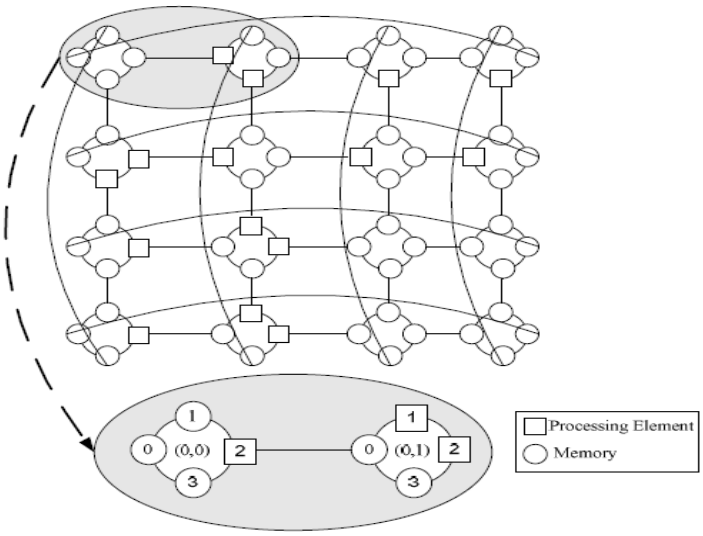


Figure 2.7. A 4 x4 hyper node torus interconnection (Khosvari et al., 2011)

Hyper node torus architecture is quite similar to K-ary n cube thus it will have the port limitation of K-ary n cube interconnection networks. While the results presented by the authors show a decreased latency, and lower message delay compared to hypercube and torus, all the results are based on a 16, and 64 nodes system, therefore no evidence exist that the proposed system will work for larger number of nodes.

P2i-torus is an architecture proposed by (Zhang & Li, 2011), it is based on torus and Plus2i architecture that is implemented by adding chords in an increasing order (power of 2) to a ring topology as it can be illustrated on figure 2.8. Plus 2 I (Liu et al., 2008) "Plus 2 I" is a special case of chordal ring with different length of chords. The p2i idea is to embed the chords into a torus network in order to minimise the network diameter, which will result in a torus interconnect with express channels and better path diversity. For instance a k-ary n cube P2i torus can be constructed by adding the rings in the k elements of k-ary n cube thus ending up with k-rings in each dimension k-ary n cube, express paths are then added into the rings.

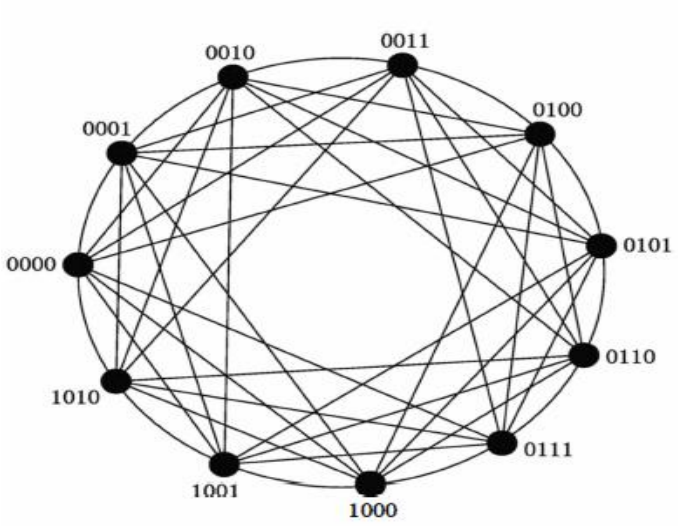


Figure 2.8. Plus 2 I ring chords (Zhang & Li, 2011)

Classic minimal routing scheme is applicable in p2i torus network with optimal routing applied on one dimension with bounding in the extra paths to allow fault tolerance. Both oblivious and adaptive routing techniques are used. In oblivious routing technique the packet that is being transmitted by the source node do not have any information regarding the other packets therefore, each packet uses a pre-calculated path to be routed. Dimensional order and load balancing oblivious routing schemes were tested in p2i torus with 4 hops saved compared to torus under dimensional order routing scheme. Thus p2i provides a shorter path compared to torus, which translates to a faster packet transmission. The features of P2i include logarithmic network diameter growth, when p2i is configured under 11 nodes as illustrated in the diagram 2.8 and linear bisectional width growth. The reason behind the decreased network diameter is the fact that p2i is constructed using a 3 dimensional torus, where the nodes are adjusted around different chordal length circles, Resulting, in a torus topology with express channels and better path diversity. The authors claim that P2i-torus can offer a throughput that is twice the channel bandwidth with minimal node degree. While the simulation results presented by the authors illustrate that p2i performs better in terms of delay compared to torus under uniform traffic distribution but performs worst on tornado traffic distribution, also the results are based on 72 core system so the scalability is not illustrated for this architecture.

Feng et al (Feng et al, 2012) proposed an interconnection network named iBT that interlaces bypass rings to a torus network. This work is quite similar to P2i architecture described above.

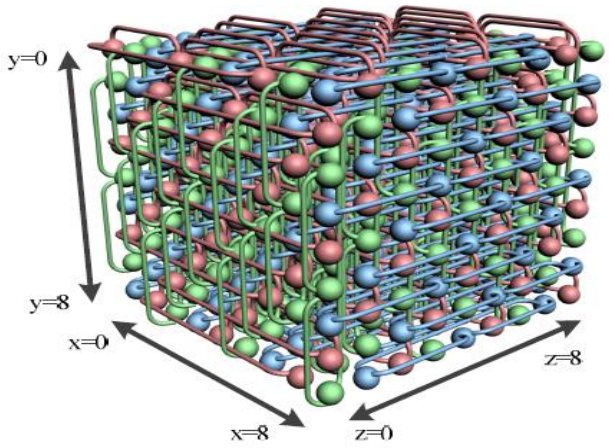


Figure 2.9. 9x9x9 3hop IBT network (Feng et al., 2012)

According to (Feng et al, 2012) A torus three dimensional network of nxnxn dimensions can construct an iBt network by adding b hop bypass links in each of the three dimensions of the torus. Therefore a 3D torus once the b-hop bypass links are added will make an iBt network that looks like a 4D torus as it can be seen on figure 2.9.

As illustrated in figure 2.10, the 6D mesh IBT network has a node degree of 12 while the 6D mesh torus network has a node degree of 10.

According to Feng "Each node in the node group will have six torus links in all +x/-x, +y/-y, and +z/-z directions, and two bypass links in one of +x/-x, +y/-y, and +z/-z directions depending on the position of the node group" (Feng, 2012).

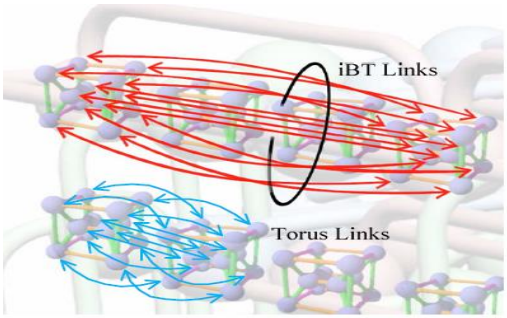


Figure 2.10. Torus and bypass links (Feng, 2012)

According to the performance analysis of the authors the IBT network performs better than 6D mesh torus or "tofu interconnect" but it is believed that it will have the same constrains in term of the network diameter that tofu (Ajima et al, 2011) interconnect has as analyzed earlier.

BlueGene Q is the third and latest generation of IBM parallel supercomputers. Compute nodes in BlueGene are interconnected using a 5 dimensional torus topology. To support the 5D torus BlueGene includes 10 bidirectional ports with an additional link added to connect the IO nodes and the compute nodes. The network supports point-to-point messages, collectives and barriers/global interrupts over the same torus. BlueGene/Q supports deterministic and dynamic routing for point to point messages, while it also supports the use of multiple routing algorithms on different packets. According to Chen "A given message always uses the same algorithm but different messages can use different algorithms (Chen et al., 2012)". This is called "zone routing" which was firstly explored in BlueGene version L but was implemented on BlueGene version Q. Packets that enter the BlueGene network are assigned one bit per direction that the packet is to be transmitted "hint bits" so that the switches can identify whether the packet needs to be routed in the minus or in the plus direction. The packets need to be routed in the long dimensions first before the switches check the bits for the smaller dimensions. The deadlocks in point-to-point are eliminated with the use of "bubble routing" or simply virtual channels. When required, the messages can change their dynamic channel, for a deterministic escape virtual channel, thus preventing possible deadlocks. BlueGene uses a dimension ordered (form longest first to shortest) deterministic routing, where waiting queues can be stored in the memory system and not in the network FIFOS (Chen et al., 2012).

Torus topologies are traditionally implemented with proprietary, costly application-specific integrated circuit (ASIC) technology. Our approach offers a simpler cost-less solution. Another disadvantage of torus interconnections is the high diameter they have. The BlueGene/Q that uses 5D torus has lower diameter than BlueGene/L with 3D torus. The problems of higher number of torus are not completely solved as synchronizing around the torus can cause a lower cycle time.

PERCS high performance interconnect is a system that was designed by IBM, in response to a high productivity, high performance supercomputers challenge (Arimilli et al., 2010). PERCS aims to improve the bisection bandwidth compared to other topologies such as fat trees interconnect and to eliminate the need for external switches. With these goals in mind, the Hub chip was used in order to support higher number of links that permit the system to be organized into a two-level direct-connect topology. Each hub chip consists of thirty one links in order to fully connect them into a star topology. Each hub chip has

communication directly to the rest of the hub ships. A supernode in Percs is achieved with the use of the chip hubs to interconnect the 32 compute nodes by combining their bandwidth thus achieving one supernode (shown on figure 2.11). PERCS support both direct and indirect routing. The direct routing supports shortest path between any two processing nodes in the system. As in the PERCS topology only two levels exist the longest direct route path can be of maximum 3 hops. Indirect routing is used to guard potential hotspots in the interconnection network.

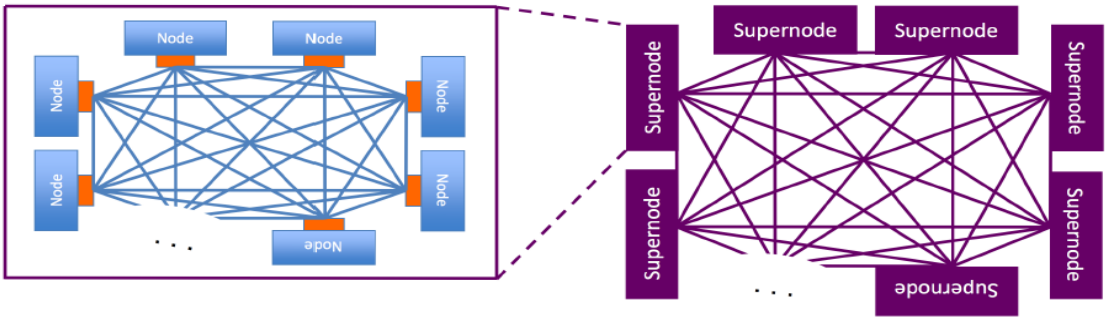


Figure 2.11. PERCS example of direct connection between nodes to form supernodes, and the connections within the supernodes previously formed (Amirilli, 2010)

IBM together with NCSA are implementing a machine name Blue Waters with the use of PERCS interconnect that will comply more than 300,000 power 7 cores. The PERCS interconnect is not believed to be a novel architecture as the use of star topology was already proposed earlier in literature (Gravenstreter & Melhem, 1998). Furthermore it can only support 32 cores to implement one supernode and even if multiple 32-nodes are connected together the complexity will be unbearable for the system.

Xmesh topology proposed by Xiao-Jing, (Xiao-Jing et al., 2007), is based on mesh but with improvements in terms of average distance by adding diagonal edges to the standard mesh topology to form some sort of a torus like topology. According to the performance analysis of the authors, xmesh performs better, by having less message delay under uniform traffic patterns compared to torus, but has higher message delay on hotspot traffic patterns.

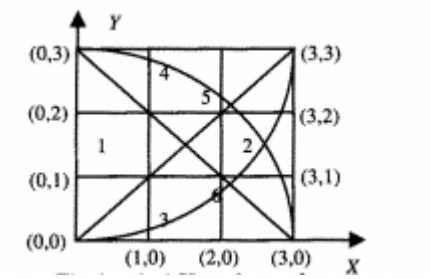


Figure.2.12. X-mesh topology (Xiao-Jing et al., 2007)

Xmesh offers an improvement over mesh topology, but the hot spot work load is high depending on the hot spots position whether they are in center or in diagonal from the network center. The reason for that is that xmesh can only support small network size that's why it performs better on uniform traffic and also the degree of the node is not equal.

Xtorus proposed by Yu hang, (Yu-Hang et al., 2012), uses the approach of xmesh topology specified above by adding diagonal mesh but this time instead of using mesh topology it's adding them to standard torus topology. Therefore It is also similar to flattened butterfly but with fewer nodes. The xtorus interconnection topology is constructed by increasing the links of a torus in diagonal and counter diagonal directions. "For *xtorus* topology at 16-node and 64-node scale, power consumption of the routers initially sensitive to the injection rate. However, after the injection rate increases to a certain point (0.1packets/node/cycle), there is almost no further increase of power consumption of the routers" (Yu-hang et al., 2012).

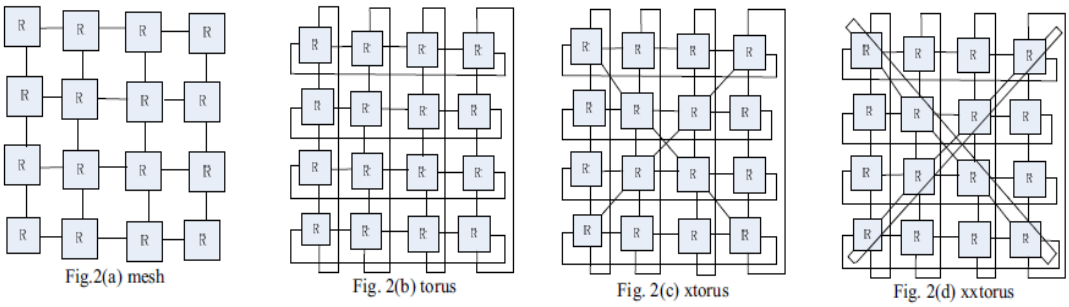


Figure 2.13. (a) Mesh (b) torus (c) xtorus and (d) xxtorus (Yu-Hang et al., 2012)

Compared to mesh, torus and xmesh, the network diameter of xtorus is shorter. In a 4x4 network the latency in xtorus is lower compared to xmesh and torus but the latency on higher level networks is not specified.

2.2.2. Indirect architectures

Fat tree networks (Leiserson, 1985) were proposed as k-ary n-tree based network topology. The only difference is that the upward links are quicker by a factor k than the downward links in order to achieve a non-changing bisection bandwidth.

Fat trees constrain is the fact that when implementing them, the switch port rates become too high near the root of the tree, thus the use of switches with the same radix and port speed is inevitable. The most suitable candidate of the fat tree topologies is k-ary n-trees as it allows the switches to be configured at all levels similarly, as illustrated in figure 2.14 a).

Interconnections based on fat tree that do not have full bisectional bandwidth are call extended fat trees such as m-ary trees (Minkenberg et al., 2011).

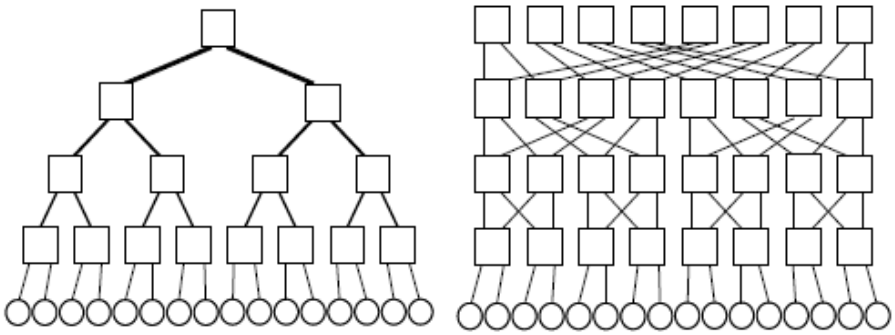


Figure 2.14. (a) Binary 4-level fat tree (b) binary 4 tree (Minkenberg et al., 2011)

Binary fat trees are based on binary trees with the only addition that their capacity is increased due to the fact that their branches become thicker from leaves to the root. A recent research undergone by (Dimakopoulos, 2011) focuses on the broadcasted scattering, the gathering of multimode and the queuing along with the total exchange of binary fat trees. This research has drawn the conclusion that binary fat trees have a number of constrains that require improvement. One of those constrains is that the multimode broadcasting of binary fat trees has excessive queuing causing unbearable delays.

Extended generalized fat tree or XGFT according to (Kariniemi & Nurmi, 2004) are interconnection networks with bidirectional multistage properties, (BMIN) which can be

extended, or scaled to accommodate different system sizes and requirements. Switches in various stages of the network have different number of bi-directional ports. Each switch is connected to the other switches using bi-directional links. Bi-directional ports are used for the connection to the bidirectional links. Unidirectional channels are used for the data transmission in the opposite direction. Bi-directional P-ports are used to connect the switch nodes to their parent nodes, while bidirectional C-ports are used to connect them to their child nodes. Each of the bidirectional C-port includes a unidirectional CUR-port and CDR-ports, and the bidirectional P-ports consist of a pair of unidirectional PDR-ports for input, and PUR-port for output. The addressing of each bidirectional C-port and P-port is in numbers which is used for identifying the parent and child nodes each port connects to.

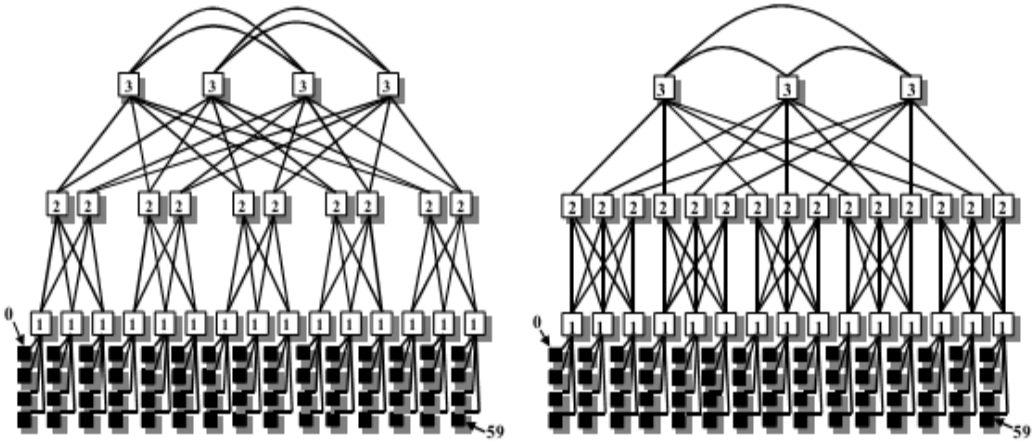


Figure 2.15. (a) XGFT(3,4,3,5,2,2,2) and (b) XGFT(3,4,3,5,3,1,2) (Kariniemi & Nurmi, 2004)

Extended generalized fat trees can be regenerated recursively to accommodate a larger system, and the connectivity along with the links used, depends on the configuration requirements, as illustrated in figure 2.15, that consists of two examples of XGFT each with different number of Routing elements in level 2 network stages but with the same number of leaf nodes. Deadlock free routing is achieved as in the XGFTs none of the routing paths can start and end at the same network node or form cycles which is quite similar to fat trees and k-ary n trees deadlock free routing approach. The routing is divided into two phases the up routing phase and the down routing phase, which they forward the packets from the up routes to the down routes only when they are routed from source to destination in order to avoid cycles in the dependency graphs. When a packet arrives to upper port is routed to the turn back channel, if the turn back channel is busy then the packet is routed to one of the parent nodes. When a packet arrives to the down port it is

routed in a downwards direction. Two routing techniques are mainly used in XGFT the original turn back when possible routing (TBWP) and the turn back (TB) which according to the table 2.1, it provides better results. Their difference is that TBWP uses address lookup table to encode and store the addresses of each node while the TB uses arithmetic addressing with address encoding for instance a node addressed as (d3,d2,d1) will be encoded as (2,2,2).

Switch type and the number of TB-channels (TBC)	Routing Algorithm	Max. Average Throughput [%]	Max. Average Latency [time slots]
Dual with 1 TBC (DUAL/1)	TB	8.18	415.0
Dual with 1 TBC (DUAL/1)	TBWP	17.8	196.7
Dual with 2 TBCs (DUAL/2)	TBWP	20.2	186.6
Dual with 3 TBCs (DUAL/3)	TBWP	22.3	175.4
Mega with no TBCs (MEGA)	TB	23.1	145.6

Table 2.1. Simulation results of XGFT with TB (Turn back routing) and TBWP (turn back when possible. (Kariniemi, 2004)

According to the simulation results of the authors the best performance and lower latency was achieved on higher number of Turn back channels (Kariniemi & Nurmi, 2006). Furthermore the configuration of the switches and the number of processing elements that can be supported is too low compared to the cost and power that a higher channel on each group of XGFT will consume. Moreover the extension of the routing algorithm proposed in XFGT (Kariniemi & Nurmi, 2006) does not provide any performance enhancements, thus the added complexity that is introduced in Turn back routing algorithm is unnecessary. Our approach expands on this work by generilising and optimising XGFT, and by providing an effective addressing scheme that uses single bits and address slicing.

Data vortex is an architecture first proposed in 1999 (Reed, 1999), then revisited in 2007 (Hawkins et al., 2007) and was extended in 2009 (Yang, 2009) to incorporate K-ary to maximise the performance by lowering the latency. Data vortex incorporates buffer-less switching and minimal routing arbitration with the use of banyan style hierarchical addressing and deflection routing. Packet contention is prevented by expanding the paths in a conventional butterfly network, thus allowing open routes to become feasible for packet deflection. Figure 2.16 (a) provides an illustration of the topology, with 3 angles,

total Height of 4 and 3 cylinders, where the paths are arranged angularly, allowing traffic to flow inwards in a spiral direction, thus going through all nodes (Hawkins et al., 2007).

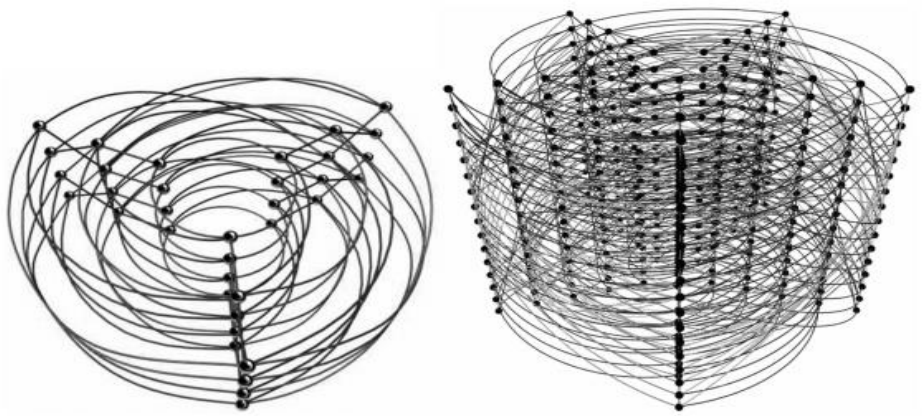


Figure 2.16. (a) Illustration of an example Data Vortex topology with three angles (b) A second example of a Data Vortex topology with 5 angles, 16 height and 5 cylinders. (Hawkins et al., 2007)

The packet routing nodes in vertex do not need any centralised arbitration as the only function by routing only one packet each time out of one of the two possible outputs, without requiring any state information or buffering. Designated slot times are used for packet injection and each packet has to fit fully in one time slot in order to be transmitted.

Extended K-ary data vortex architecture uses the same buffer less and selection routing techniques as data vortex with the use of 4-ary in an attempt to reduce the forward latency. Research studies have illustrated that even though Data vortex binary topology has higher forward latency, the overall throughput is much higher, thus a faster packet transmission is achieved (Yang, 2009).

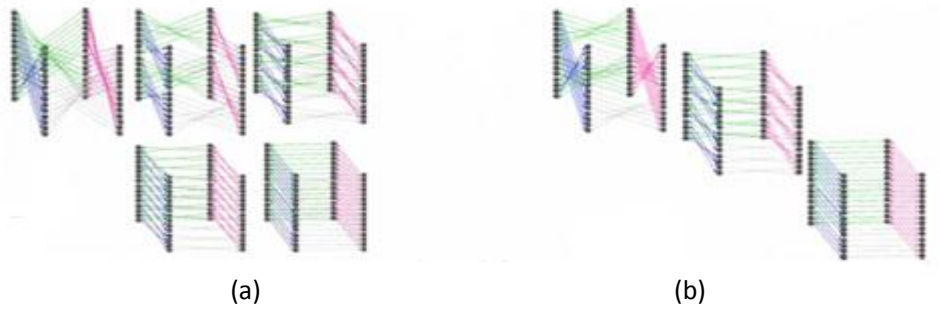


Figure 2.17. (a) Original data vortex of 4 angles, 16 height and 5 cylinders (b) extended 4-ary data vortex with 4 angles, 16 height and 3 cylinders (Yang, 2009)

Both The data vortex and k-ary data vortex interconnection networks have a various number of constrains. One of the main constrains is that the packets are injected only on one time slot. Thus the size of the packet's allowance can cause problems in large data scale applications and systems, that require packets to be injected continuously without having to wait in the queue for the next available time slot.

D. Ludovici proposed the use of high-dimensional topologies based on fat trees (Ludovici, 2009). In the same way, J. Kim investigated the use of a flattened butterfly to solve the scalability limitations of such network design architectures (Kim, 2007). Flattened butterfly architecture uses more links that other topologies under the same network scale. It exploits high radix routers in order to achieve lower cost and better performance along with higher path diversity than butterfly or k-ary n-fly interconnection networks. Flattened butterfly can be constructed by flattening or combining the routers of each row of a butterfly interconnection network. For instance a 4-ary 2 fly interconnection network consists of 8 routers or switches with each one connecting 4 inputs and 4 outputs, it can be converted into a flattened butterfly network by combining the two routers of each row to form one router thus totaling 4 instead of 8 routers. With this way channels/links that connect each row of routers together can be eliminated, thus decreasing the total cost and complexity. All the channels are bidirectional and symmetrical. The routing in flattened butterfly is simple and consists of two stages, a node needs to hop to the local router and then hop from the local router to the final destination of the packet.

Many dimensions are needed in order to be able to scale this architecture to accommodate large number of nodes which means that ending up with higher cost is inevitable. Another constrain with this solution is only suitable for high Radix routers. Moreover, as stated by R Das, (Das, 2009), all these high-dimensional on-chip networks designs suffer from traffic concentration, that causes a reduction in the maximum network throughput.

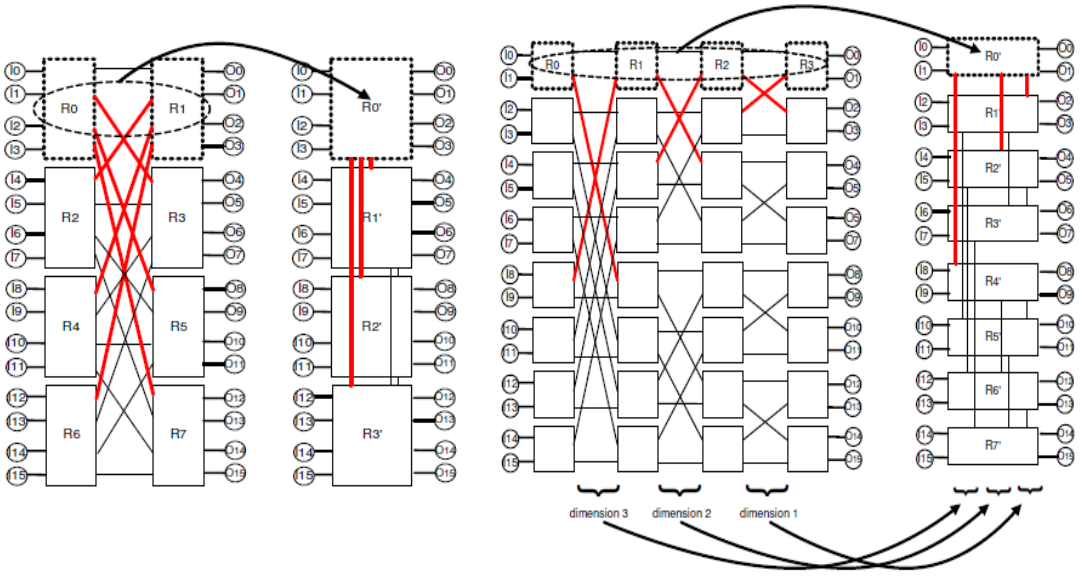


Figure 2.18. k-ary n-fly networks converted to flattened butterfly (Kim et al, 2007)

Beyond fat tree unidirectional multistage interconnection network was proposed (Gomez, 2008) in an attempt to simplify the fat trees adaptive routing and use a load balancing deterministic routing technique. By using the deterministic algorithm, selection functions and extra hardware resources for in order delivery insurance are not required, they claim that the complexity and the power consumption is reduced significantly by using a one bit look up for each traverse, compared to the traditional adaptive routing fat trees.

The deterministic routing algorithm is established on the reordering at each switch in an ascending phase by transforming the addresses from right to left.

Another simplification is the force of the packets to reach the last switch in ascending phase while the movement of packets in the descending phase is performed by direct interconnection from the last switch stage, avoiding the movement of packets through the switches. This adjustment simplifies the fat tree network as now it is considered a unidirectional multistage interconnection networks (UMIN).

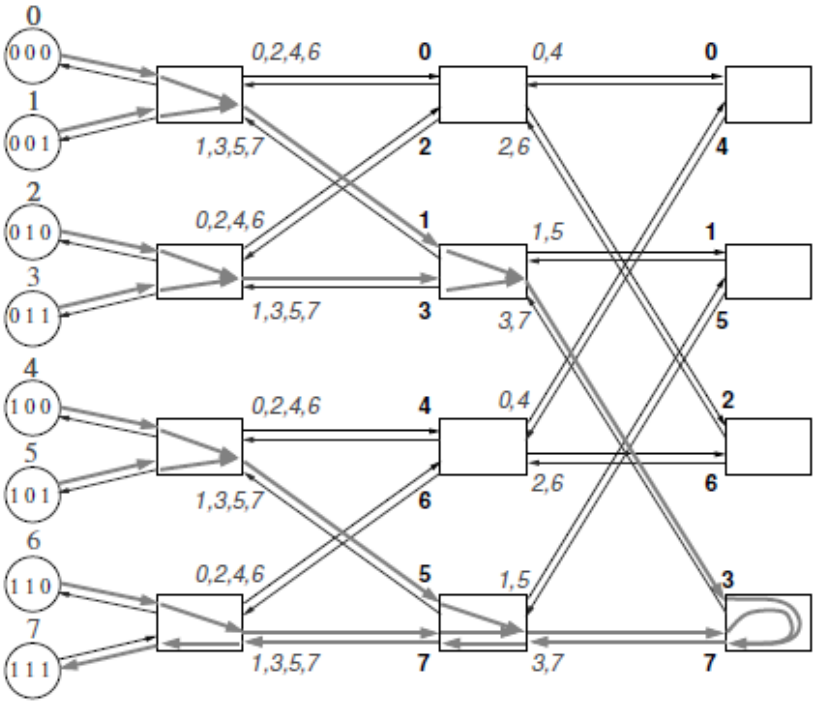


Figure 2.19. A 2-ary 4-tree with balanced deterministic routing forcing to reach the last stage of the fat-tree. Each switch port shows its reachable destinations. All the routes to node 7 have been highlighted. (Gomez, 2008)

The main constrain of this fat tree based architecture is the latency that it will have to endeavor when retrieving the full address, along with the latency when checking the lookup table for that address. Our proposed zoned node architecture can achieve far better performance results in terms transmission time as it only provides 1 bit latency per stage.

Dragonfly is a recently proposed topology by John Kim (Kim, 2008) that can be used as an alternative to the dominating fat tree topology used in infiniband platforms. The dragonfly is a hierarchical topology that reduces the network diameter and the number of long links. It includes multiple groups that are connected by all to all links, which means that at least one link is connected to each group. Each individual group can have its own topology inside, any type of topology will work but the authors recommend flattened butterfly. For deadlock prevention multiple virtual channels are required. On the fly adaptive routing was proposed as a solution for dragon fly networks in order to support exascale supercomputing by Garcia et al with the use of high radix routers. ”Dragonflies are organized as groups of routers. Links between routers can be either *local* or *global*. Routers within a group are interconnected by means of a complete graph” (Garcia, 2012).

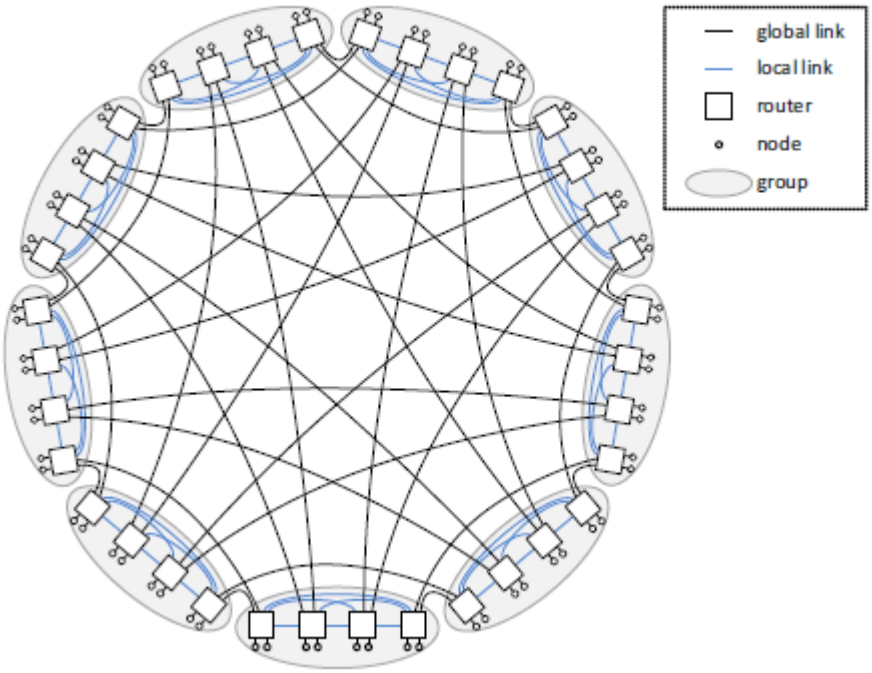


Figure 2.20. Dragon fly topology with 36 routers and 72 compute nodes (Garcia, 2012)

The simulated results of the author's research (Garcia, 2012) illustrate a good candidate for future systems. To be able to achieve the optimum behavior in dragon flies, advanced congestion look ahead is required along with non-minimal global adaptive routing which is not currently supported by any existing hardware technology.

Simplified fat tree router architecture proposed by (Bouhraoua, 2009) is a buffer-less router architecture, based on fat tree. The topology used is a Multi-Stage Interconnection networks topology (MIN), a class of bi-directionally folded MINS known as fat tree with its contention removed. A matrix of routers with n-rows and 2^ (n-1) columns, each router has 2 clients thus making the total clients of each network 2^n. Each network is repeated to form multiple groups of clients. Each client is addressed within the interval of (0, 2n-1) starting from left to right with the packet structured with the destination address field and the data. The routing in this approach is the same as the routing in binary trees, it relies on the contention elimination along the destination route, by the doubling of downward direction links. When a packet enters the network, it is forwarded in the upwards direction until it is able to reach a router that has a path to the destination node and it is then routed downwards to its destination (Bouhraoua, 2009).

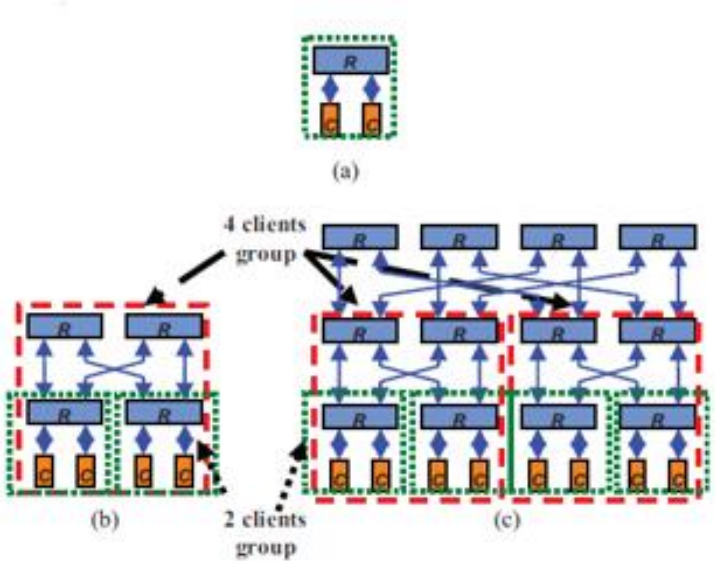


Figure 2.21. Simplified fat tree router example with: (a) 2 clients group (order 1); (b) 4 clients group made of 2 two clients groups (order 2) (c) 8 clients group made of 2 four clients groups (order 3) (Bouhraoua, 2009)

This architecture will have massive cost constrains and power consumption when a large number of cores/clients is introduced due to the fact that a large number of routers is required to support the clients. Moreover the fact that the nodes are addressed in an increasing order will cause complicated addressing mode. The routing algorithm used can cause possible deadlocks, as if a number of packets that are simultaneously transmitted, required to go to the same router for their destination path, then the traffic on that router will be too heavy. In our view this is just sub-trees or zones that are connected in a butterfly configuration, it can be easily accommodated by our proposed zoned architecture.

Tianhe 1A is a combination of CPU and GPU processing elements. It consists a total of 7168 compute nodes. The interconnection network used in THIANHE is constructed using application specific circuits (ASICs) a high radix switch and a network interface chip which is the host for the service and compute nodes (Xie, 2012). The topology used in TIANHE 1A is a hierarchical fat tree based with 11 maximum hops and 480 switches. Each switch connects 16 processing elements with electrical transmission used for communication.

The second layer includes 11 of 384 port switches as illustrated in the figure 2.22 that are connected with QSFP optical fibers.

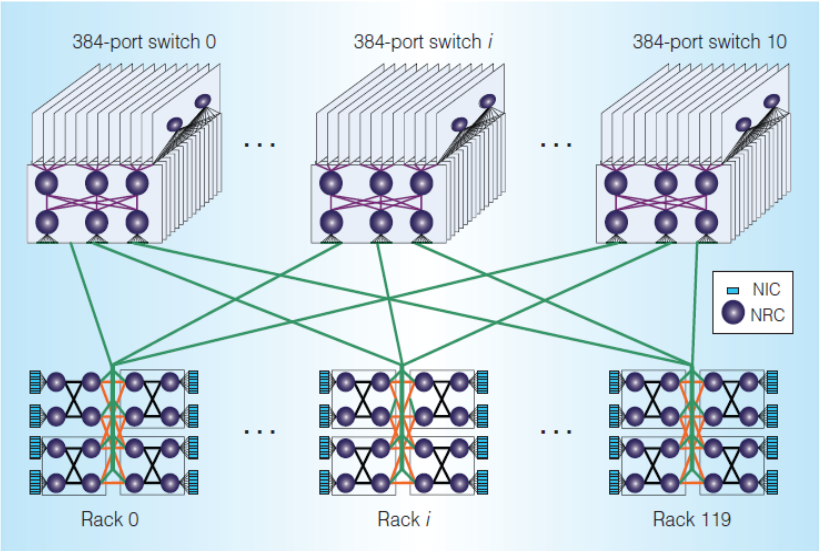


Figure 2.22. TIANHE 1A interconnection network topology (Xie, 2012)

It uses source routing and wormhole switching with the format of the packet with a total 256 bits. Virtual ports with memory-mapped registers are implemented during every user-level communication to ensure protection. Thiane 1A includes an address translation table used in each transmission to identify the destination by decrypting the address, along with a routing table to check the address destination once the decryption process is completed, which can be a time consuming process, especially in exascale supercomputing architecture that require fast transmissions.

The InfiniBand Trade Association defines the InfiniBand Archhitecture (IBA) that evolved around the virtual interface architecture (VIA) (Mellanox Inc, 2012). In simple terms it introduced a switched fabric that offers low latency and high bandwidth. The communication architecture of Infiniband is used for network based inter-process communication. Infiniband is a serial point to point full duplex interconnection network topology. Infiniband networks include a set of hosts that are connected using point to point links and switches (Bogdanski, 2013). Switches and hosts are addressed using local identifiers in each subnet. Each subnet includes a subnet manager, which initializes configures and bring up the network. Routing tables are then calculated and updated regularly to ensure performance. The process of calculating and updating the Routing table can be a timely process causing a lot of idle time in the message traverse.

Mellanox Technologies has recently extended the IBA, adding CORE-Direct capabilities to CX HCAs that manage a predefined data-dependent communication pattern posted as a single InfiniBand Multiple Work Request (MWR) and executed by the HCA. An

Infiniband switch can be used to interconnect the various devices. When two or more Infiniband switches are deployed, a 30 Gb/s interconnect is used between them (Lin, 2004).

In our solution routing information is embedded into the message allowing faster rates of traverse while in Infiniband the routing information is extracted from the switching table in each switch.

2.2.3. On-chip designs

So far we have discussed the off-chip designs which included direct and indirect networks. In this section, we briefly review some interesting multi-core on-chip interconnections.

Packet based on-chip interconnection network was proposed by Dally (Dally, 2001) as these type of networks structure the top level wires on a chip and facilitate modular design. Modularity results in enhanced control over electrical parameters and hence can result in higher performance or reduced power consumption. These interconnections can be highly effective in particular environments where most communication is local, explicit core-to-core communication. However, the cost of distant communication is high.

Piranha is a prototype architecture developed by Compaq (Barosso, 2000). This architecture exploits chip multiprocessing. It integrates eight processor cores together with two level hierarchy in a single chip, one or more memory controllers, a cache coherence protocol, one or more coherence protocol engines; and an interconnect subsystem.. “Piranha” architecture as shown in figure 2.23 consists of an intra-chip switch for the interconnection.

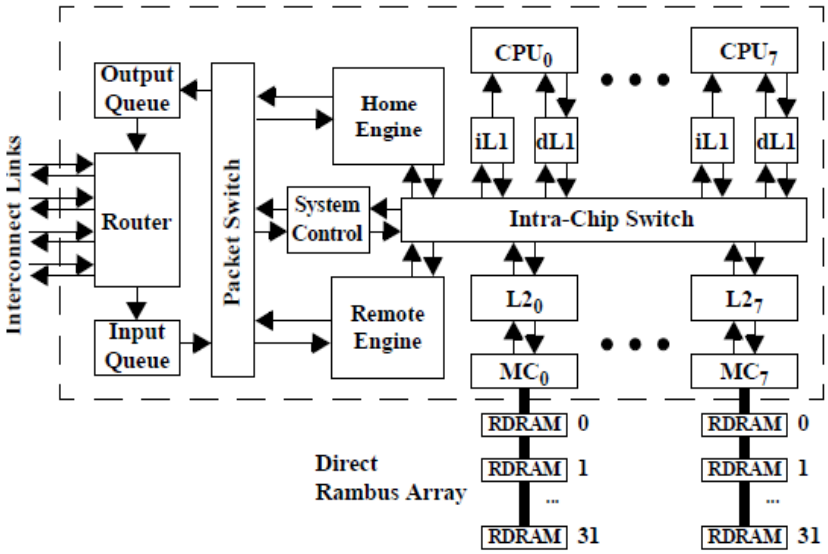


Figure 2.23. Piranha architecture (Barosso, 2000)

While Piranha is a simple and not complex solution, that can provide more than satisfactory results, it is difficult to manage the data transfers from clients effectively due to flow control and arbitration.

According to Rajeev Sivaram (Sivaram, 2002) a switch that can be used in a large parallel system must offer really high throughput whilst low latency for a various sized messages. Architectures with queuing output have proven to perform unbelievable well in throughput terms; their performance may slow down when used in systems where most of the packets are extremely small. On the other hand, architectures with input queuing offer limited throughput or require fairly complex and centralized arbitration that increases latency. A solution to that problem is given by Rajeev Sivaram (Sivaram, 2005). An architecture with High-Performance Input-Queued Switch which performance was tested under simulation and while it offers low latency despite the message size, it also offers high throughput. Furthermore, it allows simple and distributed arbitration. It uses a dynamically allocated multiqueue organization, pipelined access to multibank input buffers, and small cross-point buffers to deliver high performance.

A communication model named Columbia network architecture that aims for future multi-core on chip networks (NOC) was proposed by (Task, 2008). In this communication model the electrical subnets setup the switches in advanced of data transmission, which

is time consuming as it reserves nodes prior of sending a tear-down packet to tear off the path which will cause congestion.

Quickpath is an interconnect proposed by Intel to overcome the bottlenecks of the previous interconnect used in their architectures. It is specially optimised for multi-core processing and uses point to point interconnect between the cores. The cores in Quickpath can communicate via Input/Output hubs, individual unidirectional links are used for read and write allowing simultaneous read/write of the data (Ziakas, 2010). A full link is given in each direction directly connected to the processor, with the data being transferred serially, with only 20bits are allowed to be transferred at a given time slot the latency compared to previous Intel's architecture decreases. While the cost is lower the scalability of this architecture lacks due to the use of hubs.

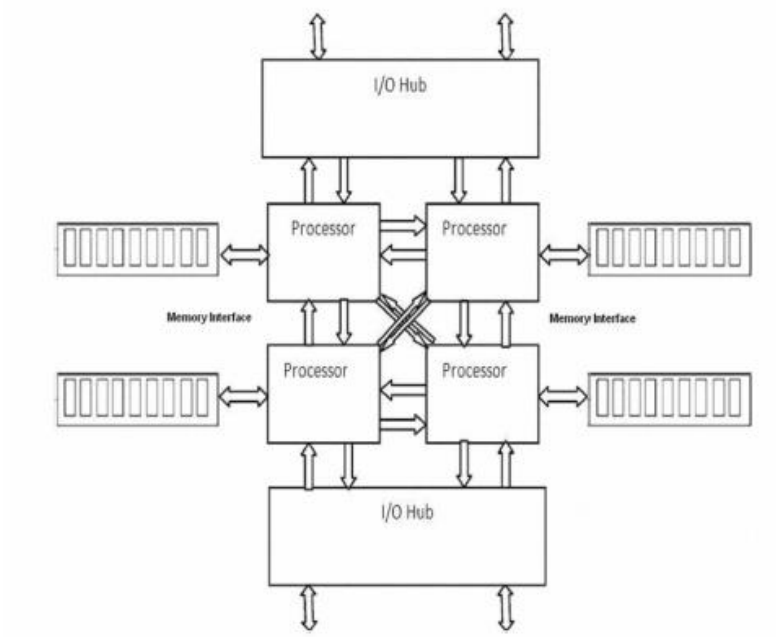


Figure 2.24. Intel quick path architecture (Khan, 2011)

Cubemach is a custom built heterogeneous architecture that uses multistage interconnection Network (MIN), making up a hierarchical on node network ONNET to cater communication bandwidth requirements. It is known in research that MIN has better latency than network on chip conventional switches. Cubemach architecture includes SCOC IP cores to reduce design turnarounds optimized per requirement at layout level local router for MIN/Globals/Subglobals (Venkateswaran, 2010). Their utilized routing algorithm, states that when a packet travels using the shortest path, then the global router equals to route and forward. The local router forwards to global and the destination check

is based on MIN algorithms where the router decides whether a path can be used based on the destination router status. This architecture can cause enormous delays in the lookup table by checking the address each time a packet is forwarded.

Hyperscalar is a dynamically reconfigurable Multi-core architecture proposed by Taiwan University (Chiu, 2010). It improves the adaptability of current CMP designs by using dynamic multi-core chips. This architecture consists of multiple single cores that can be dynamically united to generate many superscalar processors. Hyperscalar uses virtual shared register files to allow a thread that is executed in united cores to be able to logically face a uniform set of virtual shared register files. The VSR files are dynamically allocated a unique address to help each united core identify them. Each node includes a request and respond agent it includes a commit table that is used to record the status of the instructions until they are completely executed. The request and responds receives request and responds from the connected core to the corresponding core and sends the corresponding request and replies operands to their destination. Hyperscalar approach has not yet researched the possible routing problems when not-adjacent roles are working together, and the communication pattern of the united nodes needs further improvements. Furthermore the commit table the mapping table and the write phase that are required can cause high latency and congestions.

Mosart approach proposed by Candaele (Candaele, 2010) is a multiprocessor architecture for high performance and scaling with lower power consumption (Candaele, 2010). A modular multi core architecture with distributed memory organisation, scaled cores and a modular switch integrated with NoC (Network on chip) interconnect to allow arbitrary communication patterns among applications. It claims to lower the interconnect latency, the memory latency and the energy requirements and usage. A novel asynchronous communication scheme, mesh topology and bus network among the ARM nodes for faster threads processing. This approach provides claims without providing any performance results. Also it is an internal design that will not accommodate most of the application requirements.

A novel all optical router for scalable wavelength switched optical NoC (Network on chip) architecture was proposed by (Koohi, 2011). Passive routing of datastreams based on their wavelength due to WDM technique. The architecture is capable of data multicasting and low latency overheads. It consists of wavelength selective filtering and uses deterministic algorithms for routing. The destination addresses and information's is

not contained on the data packets but instead it is included in the wavelengths of the optical signal. It incorporates two different types of packet injection such as optical data streams that are targeted to corresponding processing cores along with Routing injected data in the source router. The fact that this architecture has a Routing table to save the information along with the IP blocks in passing router can cause unnecessary delays and memory usage.

Roca et al (Roca, 2011) proposed the use of a modular switch architecture that supports multiple output links with the use of individualism in the circuits as each output port is managed independently. The proposed switch is a pipelined wormhole switch where each input port can reach any output port direction. All the input ports can also reach the local port connecting the core and the switch, forcing each output port to behave as a multifunctional switch by buffering, routing and forwarding. The modular switch design requires an increase in the resources such as the buffer requirements, resulting in a higher area for the modular switch compared to the canonical switch and a higher frequency.

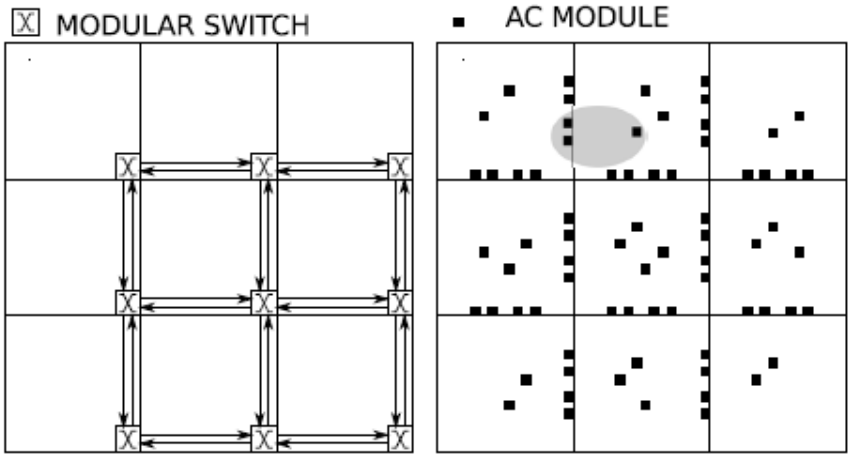


Figure 2.25. Modular switch floorplan vs distributed switch (Roca, 2011)

However in this approach the authors did not specify the configuration of the switches and nodes in a parallelized network. Moreover their proposed switches can even be applied in our proposal, to achieve even lower latency with the use of individuality in each output port both in the side links and down links.

Another problematic area of the modular switch architecture is that the individuality of each output port can increase the overall power consumption of the architecture.

Cuda or Compute unified device architecture is developed by NVIDIA and it involves the use of multi-threaded Graphic processing units to provide a many core system (Shi, 2009). Cuda on-chip interconnection can be designed in a number of ways such as butterfly, 3D torus, mesh, 2D torus, ring and crossbar (Bakhoda, 2009). CUDA uses GPUS in an attempt to maximise the performance but the latency tolerance of GPU programs has not yet been addressed directly. According to L Shi (Shi, 2009) and Bakhoda, (Bakhoda, 2009) VCUDA and vgrim that both use CUDA introduce a large amount of latency due to the fact that each Cuda call is routed to another virtual domain.

ReKonf many-core architecture that is proposed by Pal, (Pal, 2012), is a reconfigurable adaptive many core architecture that reconfigures dynamically its components by adapting to the application needs. Rekonf is constructed by reconfiguring the components in a way to mimic the properties of each different configuration of topology. Vital parameters of applications are continuously monitored. Rekonf morphs the architectures by tracking core utilisation, cache sharing of threads and live cache utilisation at runtime, while keeping execution state intact. The authors (Pal, 2012) claim that an adaptive architecture can reduce the power consumption but there is no evidence on the performance and the power reduction capabilities of that architecture in more than 256 processing cores.

Flexible heterogeneous multi-core (FMC) architecture proposed by Barcelona computing research centre, (Pericas, 2007) uses microprocessors to support both single and multi-thread processing. It is based on a novel processor micro architecture that allows the instruction window size to be changed at runtime by simply distributing the work between the multiple small cores, which can reallocate their memory as required. The FCM architecture adapts dynamically to the requirements of the threads so that reconfiguration happens dynamically without intervention of the operating system or the programmer (Pericas, 2007).

The known issues with this architecture is how to interconnect as the multiple cores use fixed delay therefore any network type can be used, but the use of fix delay can cause foreseen high latency.

2.3. Summary

In this chapter we have discussed the background and related work in the context Zoned interconnection. We have identified the main issues in the area of interconnections such as the high power consumption, low scalability along with the communication latencies and overheads of current architectures. A critical evaluation of previous work in the area of high performance architectures has been provided, along with the challenges faced by each architecture. On-chip topologies along with off-chip (direct and indirect) interconnections were fully analyzed. The background definitions provided such as delays, throughput, etc., will serve as network parameters to help us design a more efficient system. Furthermore this chapter had clearly shown the necessity towards a generalized approach to fat tree topologies, to fully address and resolve the issues that faces large scales systems. The following chapter will focus in exposing the contribution of this research by introducing and analyzing the Znode architecture.

CHAPTER 3. MODEL STRUCTURE

3.1. Introduction

This chapter introduces an architecture called the Zoned node, which is abbreviated as Znode and pronounced “ZEE” node. Several connected Znodes form a Super node in a large context. The design of Znode and Snode is based on controlled power consumption, managed network complexity, faster packet transmission with lower latency and higher throughput. A definition of the architecture is provided along with a description of the topology, and the connectivity. A general overview of the architecture, highlighting the salient features that enable the implementation of high performance machines, is presented throughout this chapter.

3.2. Topology basics

The Znode architecture has evolved from the families of folded unidirectional multi interconnection networks (UMIN) that constitute the foundation of fat tree classes introduced by Leiserson (Leiserson, 1985). Fat tree and k-ary n-tree networks have been successfully used in large supercomputer architectures for the last 20 years with slight changes in the topologies depending on the system requirements. For instance IBM and SP supercomputers (Chen, 2012) adopted such networks with modified design.

The fat tree family architectures include some interesting characteristics such as scalable bisectional bandwidth and the fact that they can be routed reaching maximum performance with a simple routing algorithm. In the original fat tree interconnection the switching nodes become larger while moving towards the top of the network or the root which is similar to binary 2-ary topology. The number of channels towards the top of the network increases as well. While the packets are rooted from source to destination processors, they are forwarded in an upward direction, until they reach the nearest common parent of both destination and source processors, they are then routed in a downwards direction until they reach their destination. This simplistic routing made fat tree variants very popular and attractive for supercomputer design.

Similarly our proposed solution consists of processors interconnected together via switches that are responsible for routing and switches that are responsible for packet injection. The Zoned can resolve the weak scalability of k-ary n-tree networks while its diameter and communication latency can be scaled due to the fact that its “height” (Network Levels) can change accordingly. This is achieved by simply adding zones at any level; of course, this requires that the routing switches have available ports to accommodate that. According to (Kariniemi, 2006) the scalability can also be achieved by XGFT architecture, however this is obtained with an increasing number of routing switches from lower to higher levels.

Znode is a hierarchical network, where the routing elements on each different level or stage can have different number of ports. It can be optimised based on the specific system requirements.

3.3. Znode concept

A pool of processors is arranged into regions called zones also known as sub-trees in the fat tree topologies. Zones are joined by a number of routing switches as we move from leaves, processors, to the root of the tree. This organisation creates a hierarchical structure known as levels illustrated in figure 3.1. This approach is similar to the way fat trees topologies are constructed.

Another new concept introduced in Znode is the stacked layers. A single Znode of figure 3.1 can be replicated many times, as shown in figure 3.2, to create layers on the top of each other, hence increasing the capacity of the system without inducing any extra latency. This comes at the price of an increase in the complexity of the topology. The layer concept distributes the traffic evenly as layers are in parallel, and only joined at the processors side. They also increase the capacity of the system allowing it to handle higher traffic load.

The full layered Znode can also be replicated several times to create a super node, abbreviated as Snode as shown in figure 3.3. This concept escalates the system scalability as we will prove in the next chapter.

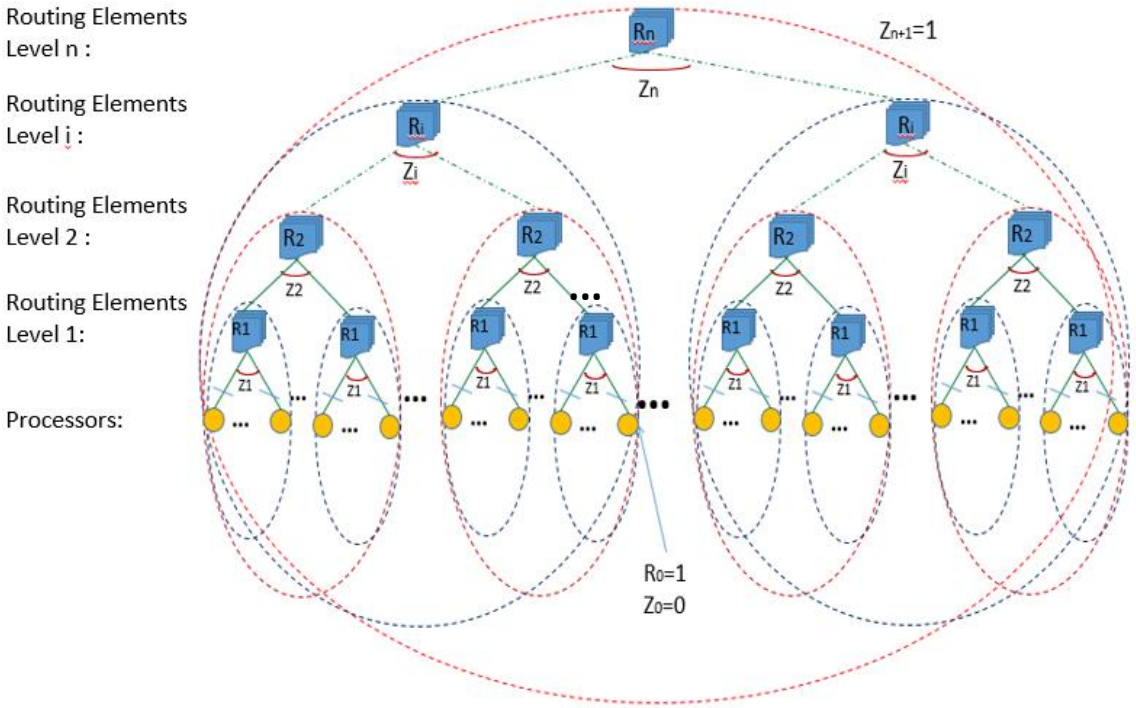


Figure 3.1. Detailed illustration of a single Znode with n levels

The number of processors in the Snode with n levels can be expressed as:

$$P = m\prod_{i=1}^{n} z_i \tag{3-1}$$

where m is the number of Znodes within a super node, and z_i is number of the zones under the level i. The total number of routing switches per level can be expressed as:

$$R_i^T = R_i \prod_{j=i+1}^{n} z_j \tag{3-2}$$

$$1 \le i \le n$$

where R_i^T is the total number of routing switches per level i, and R_i is the routing switches per zone z_i at level i.

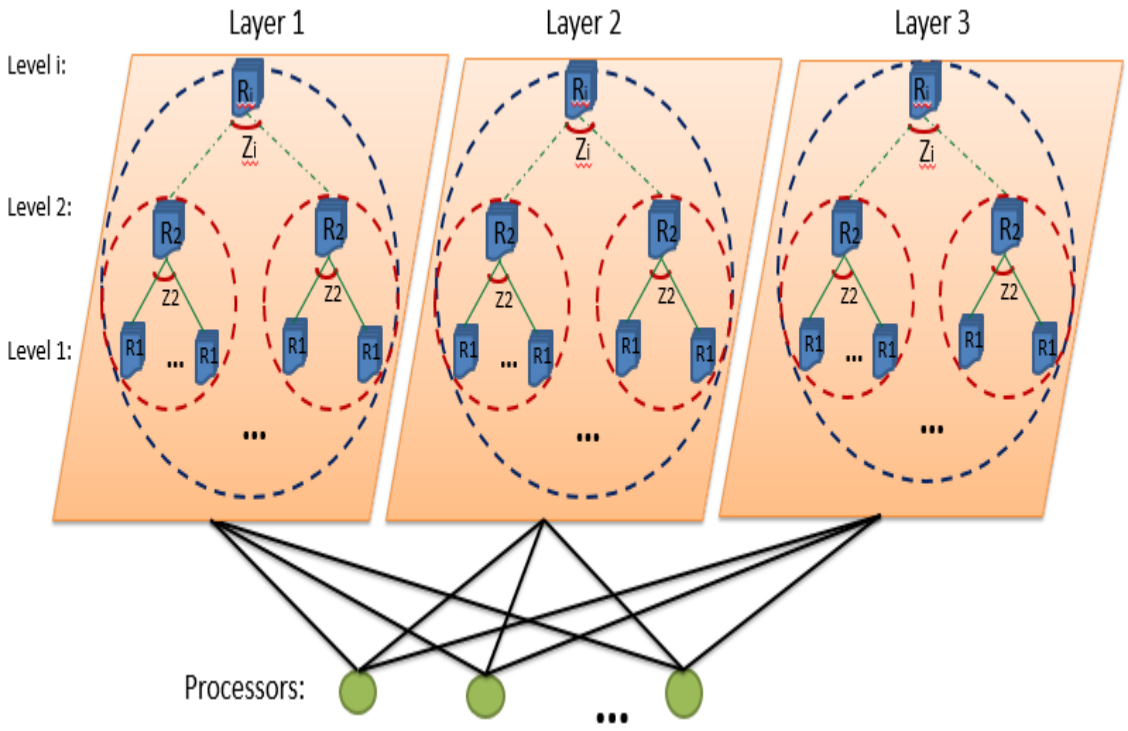


Figure 3.2. Layer structure in Znode

3.3.1. Recursive Construction of Znode and Snode.

A Zone node with n levels can be created recursively from leaf to root as follow:

$$
\begin{aligned}
& Z_1^{(k)} = (z_1, R_1^{(k)}, 1) \\
& \ldots \\
& Z_i^{(k)} = (z_i \circ Z_{i-1}^{(k)}, R_i^{(k)}, C_i^{(z)}) \\
& 1 < i \le n \quad 1 \le k \le l
\end{aligned}
\tag{3-3}
$$

$Z_i^{(k)}$ is the Znode of layers(l) up to level i. z_i is the number of zones or sub-trees under level$_i$, R_i is the number of routing switches joining the sub-trees Z_i and $C_i^{(z)}$ is the connectivity function that connects the routing elements of level i-1 to level I can be. In particular, z_1 represents the number of processing elements attached to the routing switches R_1, through point to point connection labeled as 1 in equation 3.3. In general the connectivity function can be expressed as shown in equation 3.4. It relates to the number of routing elements, the number of zones and the degree of connectivity. The degree of connectivity ψ_i refers to the extra communication links coming out of each routing switch. We will study two connectivity functions omega and butterfly enhanced with the connectivity degree ψ_i that we propose in this thesis, in the next section.

$$C_i^{(z)} = f(R_{i+1}, R_i, \psi_i, Z_i) \quad (3\text{-}4)$$

A super node, Snode, can recursively be expressed from a layered Znode as shown in equation 3.5.

$$S^{(l)} = (m \circ Z_n^{(l)}, C^{(s)}) \quad (3\text{-}5)$$

Where S is the super node constructed from replicating Znode of level n and layer (l) m times and connecting them using the connectivity function $C^{(s)}$. The connectivity function $C^{(s)}$ of the super node, in this thesis is full connectivity. Figure 3.3 illustrates a super node with 4 fully connected Znodes, at all the 4 levels. The level to level connections in the figure 3.3 simply abstracts all the connections from each level and from the corresponding routing switches of each Znode. Figure 3.5 illustrates the detailed connection between the levels and routing switches for 2 Znodes.

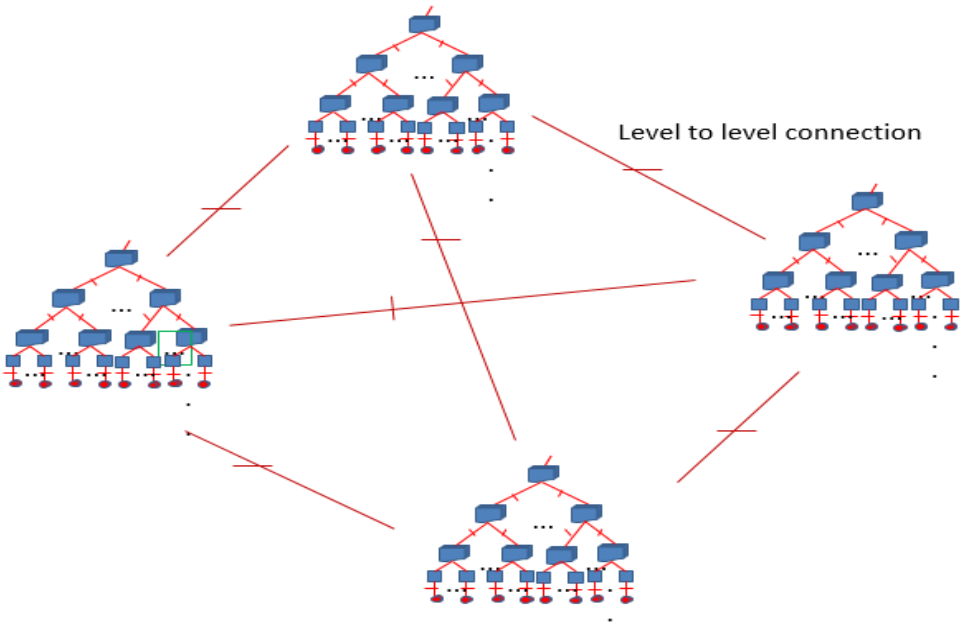


Figure 3.3. Super node of 4 Znodes within a Snode

While these efforts complete an extended multiprocessor infrastructure, interconnection enhancements are required. The Znode architecture takes advantage of the average traffic pattern, to dedicate additional links (by providing side channels) for connections with heavy traffic (near the center/down the cascaded tree traffic) and fewer links for lighter traffic (up from the cascaded tree traffic). These schemes are implemented based on

different routing techniques that will be analyzed in the next section. By studying communication patterns exhibited by a set of applications, we can gain an insight into various bottlenecks that can lead us to find ways in which the interconnection can be dynamically configured to mitigate the impact of latency on performance. In this work full connectivity is adopted to reduce the latency in the overall network unlike related works mentioned in the literature review such as Blue gene (Chen 2012) and dragonfly (Garcia 2012) which defines their interconnection schemes of different types such as hypercube etc. The connectivity function that represents the full connectivity of Super node is defined as:

$$C_i^{(s)} = h_i f(z_n) \tag{3-6}$$

Where h_i is the number of side channels, shown in figure 3.3, and detailed in figure 3.5, which involves the connections at each level i. This parameter h_i , provides a way to manage the connections between the Znodes at each level. For instance, if h_2=0, there is no connection between the Znodes at level 2 within the super node. If h_i>1 more links will be used to connect the Znodes at level i.

3.3.2. Semantics of Znode and Super node:

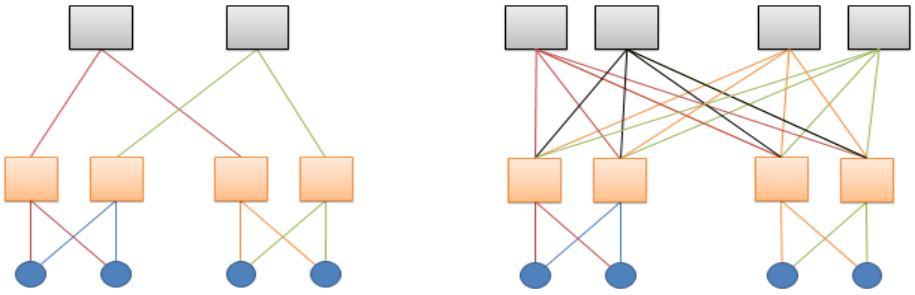
A Znode can semantically be defined from (l) layers, (n) levels, (z_i) zones, (R_i) routing, elements, and ψ_i connectivity degrees associated with $C_i^{(z)}$ connectivity functions as:

$$Z_n^{(l)} \stackrel{\text{def}}{=} \left[\frac{(R_1, R_2, \dots, R_n)^{(l)} / (\psi_1^{(C_1^{(z)})}, \psi_2^{(C_2^{(z)})}, \dots, \psi_n^{(C_n^{(z)})})}{z_1, z_2, \dots, z_n} \right] \tag{3-7}$$

A super node of m Znodes can be defined as $S_m \stackrel{\text{def}}{=} m \circ Z_n^{(l)}$, which is a replication of Znode m-times. The $Z_n^{(l)}$ notation represents a single class of Znodes with n levels and l layers, but the underlying specifications on the number of zones (Z_i), routing elements (R_i), communication links (ψ_i), and the connectivity functions will determine the detailed configuration of it. The superscript (l) layers defines the stacking of the parallel layers all with the same number of routing switches. The superscript ($C_i^{(z)}$) connectivity function determines the patterns of connectivity at level i and level i-1 using the degree of connectivity (ψ_i). The optimal Znode that can be constructed with the parameters that

minimise the power consumption and the message delay in the network is labelled as $oZ_n^{(l)}$.

An example of a single Znode with 4 processors and 2 levels, Z_2, represented in figure 3.4(a), can be defined by $Z_2 \stackrel{\text{def}}{=} \frac{(2,2)/(1,1)}{(2,2)}$. The topology shown in 3.4(b) can be represented as $Z_2 \stackrel{\text{def}}{=} \frac{(2,4)/(1,2)}{(2,2)}$ where the degree of connectivity at level 2 is equal to 2. Figure 3.5 also shows a super node with 2 Znodes, 32 processors and 4 levels; The Znode for that example can be represented as $Z_4 \stackrel{\text{def}}{=} \left[\frac{((1,2,2,2)/(1,1)}{(2,2,2,2)}\right]$ but since 2 Znode exist the super node will be represented as $S_2 \stackrel{\text{def}}{=} 2 \circ Z_4$, therefore the final representation would be : $S_2 \stackrel{\text{def}}{=} (2) \circ \left[\frac{((1,2,2,2)/(1,1)}{(2,2,2,2)}\right]$. The degree of the connectivity function $C_1^{(s)}$ in this case embodies a fully connected network. Notice that the layer and the connectivity exponents are not shown in these examples. A layer equals to 1 is omitted from the semantic expression for convenience. The connectivity functions are also omitted, because they apply equally to all levels and in these examples we have chosen the butterfly pattern. In general one can use a symbol to represent the connectivity functions in the expression. For instance when $C_1^{(z)}$ and $C_2^{(z)}$ are of butterfly pattern it can be expressed by replacing the letter C with (b) to identify the bufferfly pattern, or (Ω) for Omega pattern considered in this work.



(a) (b)

Figure 3.4. Representation of butterfly connectivity function (a) m=1, n=2, ψ1=1, ψ2=1, zI=2, z2=2, R1=2, R2=2. (b) m=1, n=2, ψ1=1, ψ2=2, z1=2, z2=2, R1=2, R2=4

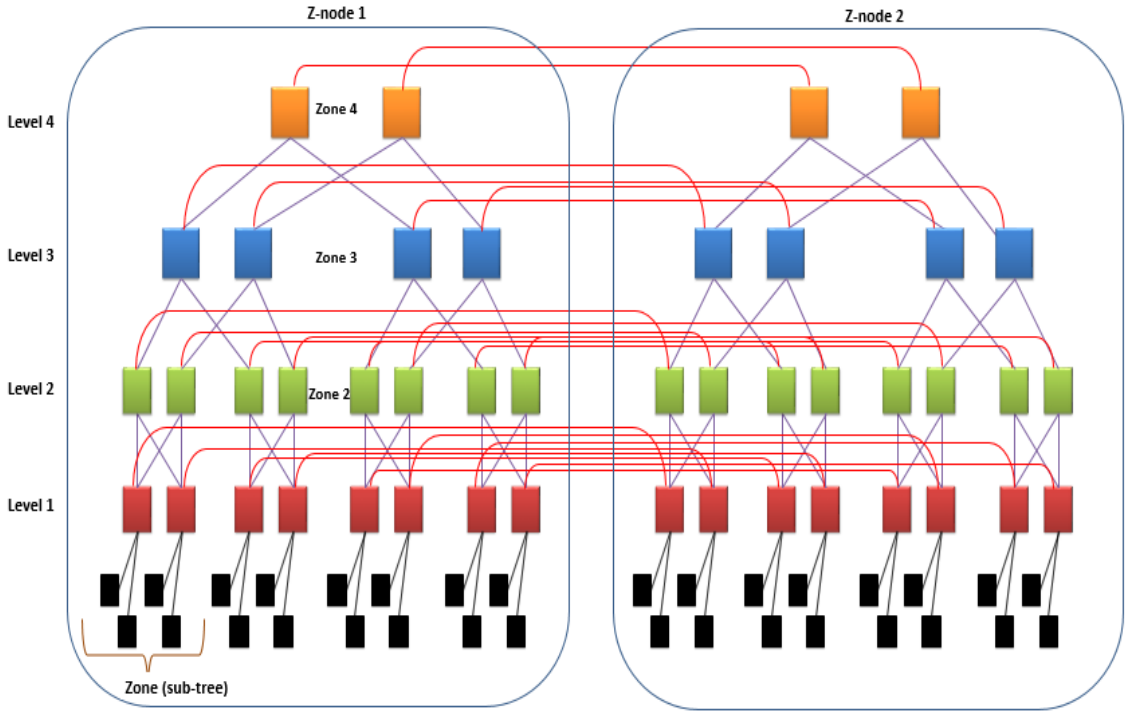


Figure 3.5. Illustrating a Snode with 2 Znodes, with h1=h2=h3=1 meaning that all the levels are fully connected

An example which includes layer >1 is shown below; in this case two levels exist along with 2 layers, which is graphically illustrated in figure 3.6, $Z_2^{(2)} \stackrel{\text{def}}{=} \left[\frac{((2,2,)^{(2)}/(1,1)}{(2,2)}\right]$. Furthermore, the k-ary n-tree defined by $P = k^n$ (Petrini, 1997) can semantically be represented by the Znode as $Z_n = \left[\frac{(1,k,k^2,\ldots,k^n)/(1,1\ldots1)}{(k,k,\ldots,k)}\right]$. Furthermore, the extended generalised fat tree XGFT topologies (Kariniemi & Nurmi, 2004) defined as $(h, 0, m_1, \ldots, m_h, w_1, \ldots, w_h)\ 1 < i < h$ where h is the height (level n), m the descendant and w the root of the sub tree with a number of processors $P = m_i * \ldots * m_h$ can be represented as $Z_n = \left[\frac{(1, w_2 * R_1, \ldots, w_n * R_{n-1})/(1,1\ldots,1)}{(z_1, z_2, \ldots, z_n)}\right]$. This shows that to increase the bandwidth in the upper levels – requirements of the fat trees, the XGFT has to provide more routing elements $w_i * R_i$ this is balanced in the Znode by providing more degrees of connectivity, or a trade-off between the number of routing switches and the connectivity degree.

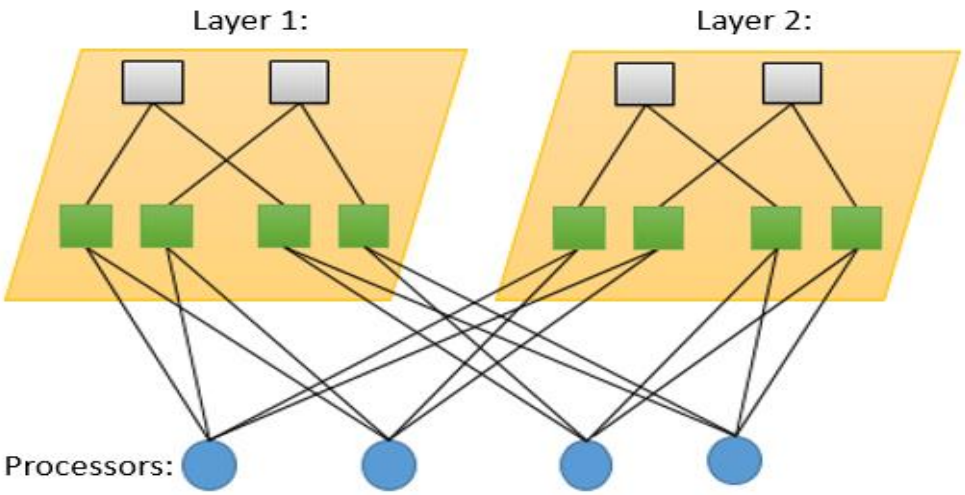


Figure 3.6. A representation of m=1, layer=2, level=2 Z1=2, Z2=2, R1=2, R2=2, ψ1=1, ψ2=1,

3.4. Connectivity in Znode

The connectivity in Znode is defined by the connectivity function $C_i^{(z)}$ firstly introduced in section 3.3. This function can implement a wide range of permutations between the routing switches at level i and level i+1. Two popular permutations are considered in this work, Butterfly (Grammatikakis, 2001) adopted by k-ary n-tree and Omega (Yang, 2000) adopted by XFGT. Our results will show that there is no performance benefit in using either of those patterns. The reason behind adopting those two patterns Omega and butterfly is purely based on the ground of comparison, to allow us to evaluate our proposed architecture against k-ary n-tree and XGFT and others alike. Furthermore, we have amended the patterns of the Butterfly and Omega by introducing the connectivity degree ψ. This shows that we can obtain better performance with less number of routing switches compared to the mentioned schemes. In this section we provide our own definition based on the degree of connectivity for routing connectivity function of increased Butterfly and increased omega connections.

The fatness of the Znode topology can be adjusted by changing the values of the routing elements in consecutive levels i and i+1 respectively. If the ratio of Ri+1 to Ri or Ri to Ri+1 is not integer, then full connectivity between levels i and i+1 is achieved (figure 3.7). Otherwise forward (positive) or backward (negative) connectivity can be identified. With the forward connectivity (ψ_i^+) the number of routing switches and hence the links

increases from lower level to upper levels, whereas with backward connectivity (ψ_i^-) the number of routing switches decreases as we move from higher level to lower levels. Our generalized Znode configuration as shown in figure 3.7 categorises FAT-trees, GFT, K-ary N-tree, XFGT, optimal Znode, power of two Znode, fully connected, and forward and backward connections.

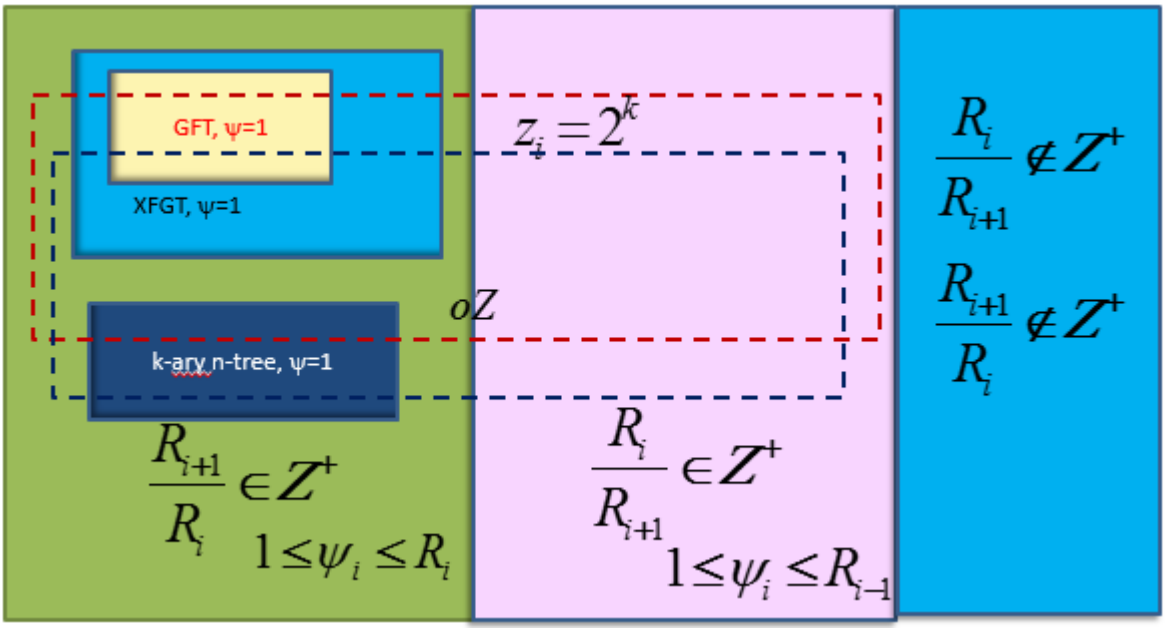


Figure.3.7. Illustrating the generalisation of the Znode structure

3.4.1. Forward (positive) connectivity

The forward connectivity between the levels is used when the routing elements in level i+1, "Ri+1" are divided into groups, such that the number of routing elements in each group is equal to the number of routing elements Ri in the zone at level i, see figure 3.8. The number of groups G_i^+ is represented as $\boldsymbol{G_i^+ = \frac{R_{i+1}}{R_i} \geq 1}$. When the number of Routing elements are higher in the top level and less in the lower levels the connectivity degree is denoted as ψ_i^+. It is bounded to $\boldsymbol{1 \leq \psi_{i+1}^+ \leq R_i}$, and the number of upper links can therefore be expressed as $\boldsymbol{\frac{R_{i+1}\psi_{i+1}}{R_i} \geq 1}$.

In figure 3.4 (b), the number of upper links per routing switch is $\left(\frac{4}{2}\right) * 2 = 4$.

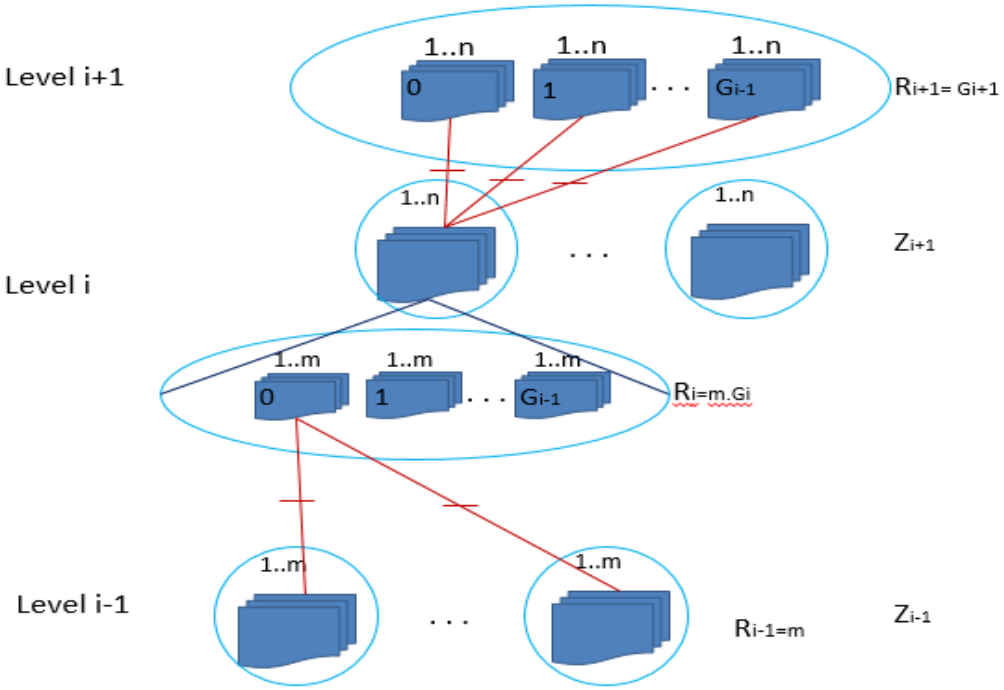


Figure 3.8. Forward Positive cyclic connectivity between levels

When the forward butterfly pattern is used, a routing element from the above level i+1 is connected to a routing element at the current level i with a connectivity function defined by the following equation:

$$C_i^z : x = (j)_{i+1,g,z_{i+1}} \mapsto y = ((j+k-1)\%R_i)_{i,z_i}$$
$$1 \le k \le \psi_{i+1}^{+}$$
$$0 \le j < R_i$$
$$0 \le g < G_{i+1} \qquad (3\text{-}8)$$

When the forward omega pattern is chosen on the other hand the connectivity function changes as follow:

$$C_i : x = (j)_{i,z_i} \mapsto y = ((jG_i^{+} + k)\%R_{i+1})_{i+1,z_{i+1}}$$
$$0 \le k \le G_i^{+}\psi_{i+1}^{+} - 1$$
$$1 \le \psi_{i+1}^{+} \le R_i$$
$$0 \le j < R_i \qquad (3\text{-}9)$$

3.4.2. Backward negative connectivity

The backward (negative) connectivity between the levels is the opposite of the forward connectivity described above. Instead of considering the groups at the level i+1, we determine the groups at level I (figure 3.9).

We define the routing ratio for negative connectivity as $G_i^- = \frac{R_i}{R_{i+1}} \geq 1$. When the routing elements in the lowest level are higher than the routing elements of the upper level the connectivity degree is denoted as ψ_i^- which is bounded to $1 \leq \psi_i^- \leq R_i$ and the routing elements R_i are divided into groups, such that the number of routing elements in the each group at level i is equal to the number of routing elements in the zone at the level i+1, R_{i+1}. The backward butterfly connectivity function can be expressed as:

$$
\begin{aligned}
&C_{i+1}^z : x = (j)_{i,g,z_i} \mapsto y = ((j+k-1)\%R_{i+1})_{i+1,z_{i+1}} \\
&1 \leq k \leq \psi_{i+1}^- - 1 \\
&0 \leq j \leq R_{i+1} \\
&1 \leq g < G_i^-
\end{aligned}
\qquad (3\text{-}10)
$$

For Omega backward connectivity we can define its notation as follow, and an example of the notation can be seen below for k=3.

$$
\begin{aligned}
&C_i^z : x = (j)_{i+1,z_{i+1}} \mapsto y = ((jG_{i+1}^- + k)\%R_i)_{i,z_i} \\
&0 \leq k \leq G_{i+1}^- \psi_{i+1}^- - 1 \quad 1 \leq \psi_{i+1}^- \leq R_i \\
&0 \leq j < R_{i+1}
\end{aligned}
\qquad (3\text{-}11)
$$

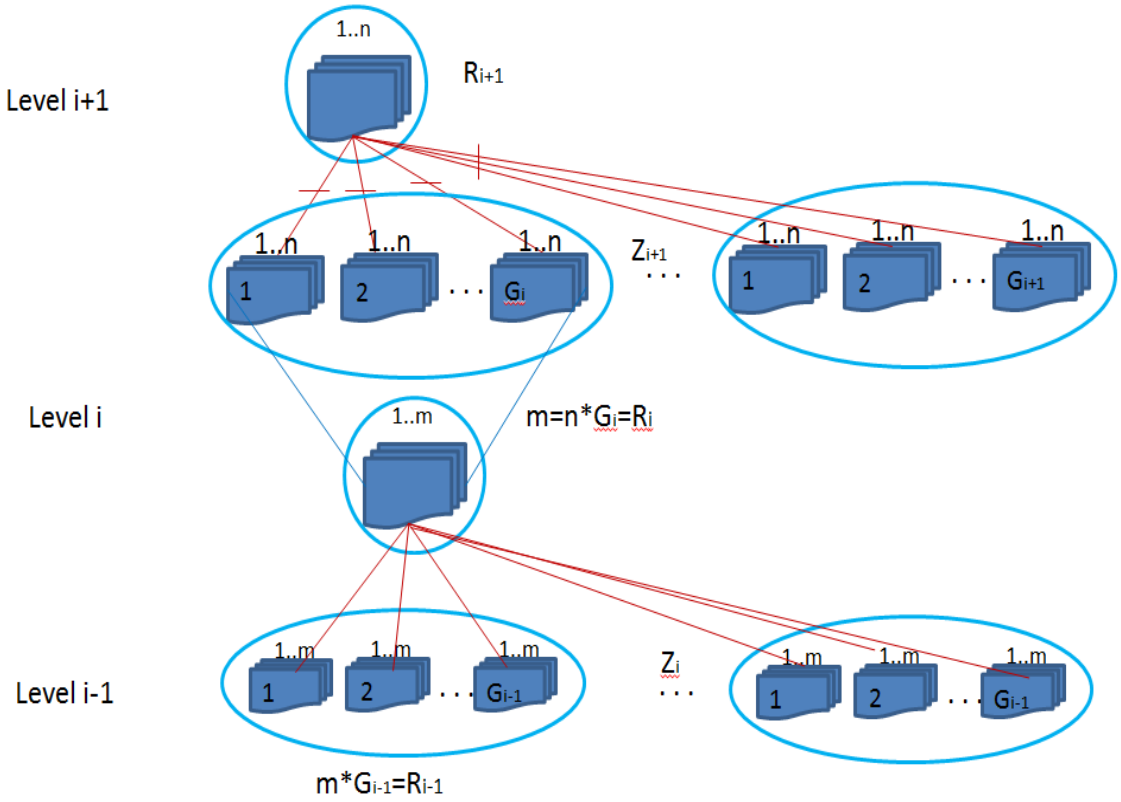


Figure 3.9. Backward Negative connectivity between levels

3.4.3. Full connectivity

Full connectivity can be utilised for both forward and backward connectivity as long as the routing ratios of the routing switches in two consecutive levels is not an integer.

3.4.4. Communication links node

Znode architecture offers several degrees of connectivity as explained in section 3.3. Each of the links configurations can be used according to the system and applications requirements. It needs to be taken into account that the higher the number of links used, the higher the complexity and the overall cost of the system would be. The degree of connectivity ψ_i can vary in each level. For instance in a three level configuration , level two can have 8 links per switching node while level 3 can have 16 links per switching node and vice versa. This feature permits the configuration to have different links per levels, which for some applications that exhibit higher traffic at certain levels radix would be an appropriate choice to reduce delay and maximise the throughout.

3.4.4.1. Number of Downlinks

The number of down links in backward connectivity can be evaluated as

$$L_i^{d-} = \psi_i^- z_i G_i^- \quad (3\text{-}12)$$

This expression shows how backward Znode increases its bandwidth for the downlink routing. In the flat tree configuration, as we will explain in the next chapter, the down link routing is usually deterministic. This means that traffics to destinations follow predefined paths without adapting. In other k-ary n-tree and XGFT the downlink is equal to z_i only. In our case, we have increased it topologically by a factor of $\psi_i^- G_i^-$. This provides alternative paths to deterministic routing all ending at the required destination. We will show in the chapter 5 the simulation results that support this claim.

The number of downlinks in the forward connectivity is

$$L_i^{d+} = \psi_i^+ z_i \quad (3\text{-}13)$$

On the other hand, for forward connectivity, the number of paths for the deterministic routing is limited. The inclusion of the layers provides additional paths at the expense of the complexity.

3.4.4.2. Upper links connectivity

The number of upper links with the backward connectivity is simply equal to the degree of connectivity.

$$L_i^{u-} = \psi_{i+1}^- \quad (3\text{-}14)$$

This provides a reduced bandwidth for the adaptive routing and hence additional layers would supply more paths to reduce the message delay and increase the throughput. However,

the number of upper links with the forward connectivity is:

$$L_i^{u+} = G_i^+ \psi_{i+1}^+ \quad (3\text{-}15)$$

Unlike K-ary n-tree and XGFT and other fat-tree topology, The Znode increase the up bandwidth of each routing switch by a factor of ψ_{i+1}^+ which can extend up to R_i.

Therefore the total number of links per routing switch can be expressed as:

$$L_i^+ = \psi_i^+ z_i + G_i^+ \psi_{i+1}^+ = \psi_i^+ z_i + \frac{R_{i+1}}{R_i} \psi_{i+1}^+$$

$$L_i^- = \frac{R_{i-1}}{R_i} \psi_i^- Z_i + \psi_{i+1}^- \quad (3\text{-}16)$$

Where L_i^+ is the total number of links for forward positive connectivity and L_i^- is the total number of links for backward negative connectivity.

3.5. Summary

In this chapter we have exposed the contribution of this research from all aspects through the overview of Znode architecture, the topological advantages and features along with the connectivity utilised. In the Znode, we have introduced several attributes that were not in any other fat tree class's work before. Firstly, a generalised semantic representation that specifies fat tree classes was proposed. This specification can be extended to super node connections, where modern architectures such as BlueGene (Chen 2012), DragonFly (Garcia 2012) and others can be identified. Secondly, the notions of layers that initiate more bandwidth and balance the traffic equally among planes have been introduced. Thirdly, the degree of connectivity that provides more paths for deterministic as well as adaptive routing was also proposed. This feature allows the configuration to increase bandwidth without increasing the number of routing switches. Fourthly, super node extensions to the Znode have introduced, this allows the Znode to scale without major upheaval to the performance, at the expense of additional side links. This topology concept is found in the new architectures, but our proposal maintain the same philosophy of the Znode creation. Several configurations were analysed such as fully-connected, forward and backward connectivity. The next chapter will focus in presenting the Routing addressing algorithm utilised in Znode along with an analysis of our proposed optimisation tool.

CHAPTER 4. ROUTING AND OPTIMISATION

4.1. Introduction

In the previous chapter we have illustrated the main features and characteristics of Znode architecture. This chapter serves as a continuation and proposes a generic routing algorithm that can be applied to Znode, and other similar fat tree architectures. The Routing algorithm is named “Sliced Routing” algorithm due to the fact that it uses sliced port labels to identify the downwards direction rather than carrying the full address. Throughout this chapter we will also illustrate the optimisation model for fat tree topologies. The parameters will be explained along with the objective function and constrains that make the optimisation successful in minimising the power without inducing extra latency.

4.2. Routing

Each routing algorithm has different features; some are more suitable for certain types of interconnects. The features are their ability to adapt to network state when making their routing decisions, also the place where the routing decisions are made along with the implementation of the routing algorithms. Routing algorithms decisions with in the routing elements can be implemented with lookup tables or with arithmetic operations. In the table look up approach, like Infiniband (Mellanox Inc, 2013) and Internet networks (Hura, 2001). Once the packet arrives at the router, the routing table is checked to find the best possible path. The lookup tables are usually generated inside the routing element at the configuration state of the network. Table lookup routing is preferred in irregular topologies such as Myrinet and Autonet (Boden 1995).

In Arithmetic operations routing, the message carries routing decisions that might be checked along its path such as k-ary n-tree and XGFT or devised before the departure of the message at the source. Arithmetic routing is preferred in regular topologies, where the state of the network is predefined. We have suggested a sliced routing algorithm,

explained in the following sections, for the Znode which is a source routing algorithm. Unlike K-ary and the XGFT, it requires no addresses in the routing switches.

4.2.1. Arithmetic Routing in Znode

The use of source arithmetic routing is preferred compared to table lookup routing, due to the fact that the latency caused by the lookup time and the time needed to save the node information is sometimes undesirable for performance computing.

Being based on the fat-tree philosophy, the Znode implements both adaptive routing for the up traffic and deterministic routing for down traffic. Adaptive routing scheme is generally preferred, as in the case of unforeseen circumstances, such as a network fault or a network change occurs, the message can still reach its destination successfully. Adaptive routing scheme can fall under one of the following categories: Progressive or backtracking; the difference is that in backtracking the messages are routed both forwards and backwards, while in progressive routing they only move forwards. In progressive routing, adopted by Znode, the message flit or header is followed by the payload, while in backtracking the header is moved further ahead of the rest of the message.

4.2.2. Adaptive routing in Znode

The upwards routing in Znode is a fully adaptive routing scheme. When a message arrives at a routing element, it is forwarded to any available uplink. Znode configuration with higher connectivity degree at the expense of the complexity can offer more upwards links. In the Snode messages are forwarded using the side links. This routing is based on hierarchical-address routing which is similar to dimension order routing algorithm (Montanana, 2009). However there are no dimensions in the Snode, full connectivity requires one part of the message address. Messages use the parts of the destination address to follow their paths to the destination, as explained in the addressing section of this chapter. This algorithm requires the routing switches to have their own local addresses referring to the identity of the Znode to be able to perform routing decisions.

4.2.3. Deterministic routing in Znode topology

The downwards routing in Znode is of deterministic nature. The message header includes portions of the encoded address, which determines the routing paths for the destination.

In fact the address corresponds to the port identifier of each routing switch at each level. Hence, side-stepping the need to have local addresses per routing switch. The lowest level routing element (level 1) is the one responsible to route messages to its destination processing node (level 0).

4.2.4. Routing algorithms in Znode topology

The combination of the three routing algorithms mentioned above, adaptive, hierarchical-address, and deterministic routing will allow a message in Znode to move first adaptively from lower levels to the common ancestor level. It then moves using hierarchical-address with the Znode identifier sidewise to the common peer level in the destination Znode. Finally, deterministically the message makes it way down to the destination processor using the port labels.

As an example of the routing algorithm used in Znode, consider a configuration with 4 zones and a single Znode where processor *p* wants to send a message to processor *q* as illustrated in the figure 4.1. The two processors have a common ancestor, which is in level 3, explained in definition 4 of the addressing section 4.3.

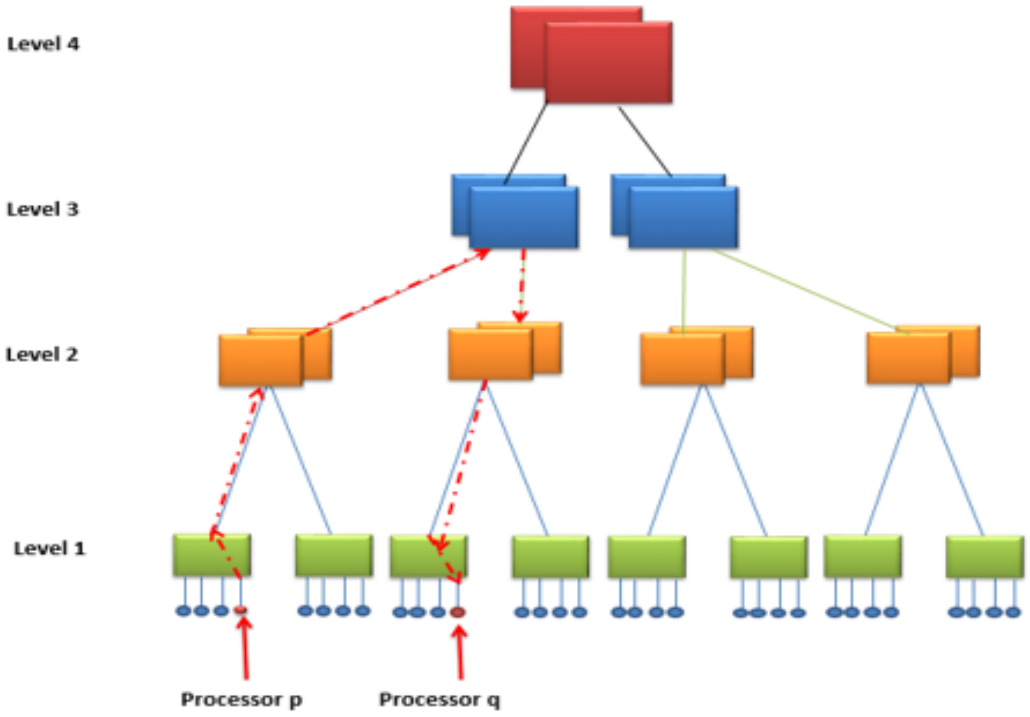


Figure 4.1. An example of routing when a common ancestor is found.

4.3. Addressing of the Znode and Snode

The neighbours of a processor *P* are those processors immediately connected to the same switching element of that processor. Each neighbour of *P* is at a distance 1 from *P*. Further distances are defined recursively as analysed below.

Definition 1: Two processors *Q* and *P* are uniquely identified by the tuples as follow:

$$Q = (q_{n+1}; q_n, q_{n-1},..., q_i,..., q_1)$$
$$P = (p_{n+1}; p_n, p_{n-1},..., p_i,..., p_1)$$
$$where \quad p_i \in \{0,1,..., z_i - 1\} \; and \; P_{n+1} \in \{0,1,..., m-1\}$$

An example can be viewed in figure 4.2, where the addressing for all the processors is based on definition 1. Processor 2 will have an address of (0;002) as is on Znode 0 under the port identifiers for 0,0,2 for levels 3,2 and 1 consecutively.

Definition 2: Two processors *Q* and *P* have a common level i in the Znode if and only if

$$q_j = p_j \quad for \; \forall j \quad i < j \le n$$
$$q_{n+1} = p_{n+1}$$
$$If \; q_j \neq p_j \quad for \; \forall j \quad then \; i = n$$

This shows how messages are routed down when they reach their common level. In the fat-tree the common switch is reached since the destination address is compared to the switch address. As illustrated in figure 4.2, processors 2 and processor 3 will have the addresses (0;002) and (0;003) consecutively, thus the common level is level 1 due to the fact that all the port identifiers are the same except the last part 2, 3 which represent the port for each processor at switch on level 1.

Definition 3: Two processors *Q* and *P* have a peer level i in the Snode if and only if

$$q_j = p_j \ \ for \ \forall j \quad i < j \leq n$$
$$q_{n+1} \neq p_{n+1}$$
$$If \ \ q_j \neq p_j \ \ for \ \forall j \quad then \ i = n$$

This definition indicates how messages are routed between Znodes in a Snode configuration. An example of a peer level is represented in figure 4.2, processor 2 (0;002) and processor 16 (1;000) will have level 1 as their peer level as their port identifiers are the same after level 1, but each processor exists on a different Znode. If they were on the same Znode then this would have been a common level.

Definition 4: Two processors *Q* and *P* in the Znode of n levels having a common or a peer level i, will be separated by the following bit directions.

$$F_i = (f_1, f_2, ..., f_{i-1}, f_i)$$
$$where \ \ f_i = q_{i+1}...q_n \otimes p_{i+1}...p_n$$
$$if \ \ i = n \Rightarrow f_n = 0$$
$$where \ x \oplus y = \begin{cases} 0 \ if \ \ x = y \\ 1 \ if \ \ x \neq y \end{cases}$$

For further analysis on how the bit directions are identified please refer to section 4.3.1.4 and for the implementation of definition 4 refer to chapter 6 section 6.4.2.

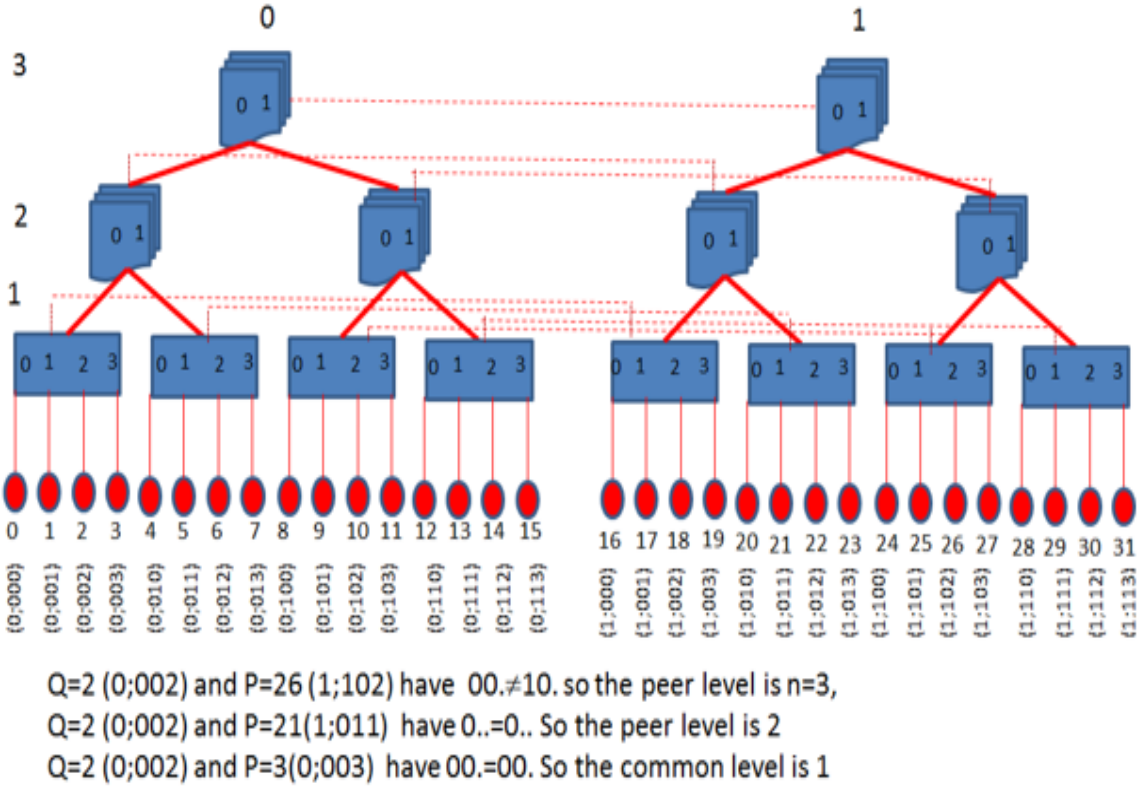


Figure 4.2. Addressing representation

4.3.1. Type of routing and address overhead

This section provides the explanation of each addressing and routing scheme utilised in Znode and Snode.

4.3.1.1. Flat addressing

Flat addressing is an addressing scheme mainly used in banyan multistage networks and inverse banyan such as omega network (Youssef, 1992). Messages are routed all the way to the top level or root switch and then routed down deterministically towards the destination processor. For systems with large number of levels (i) the flat addressing would have caused enormous latency. For instance in a configuration of 8 levels,(Z_8^1) when a packet is transmitted from a level 1 node, and the destination is a level 4 node, the packet will have to be routed all the way up to level 8 switch before it starts the down-routing. The downwards routing in flat addressing scheme follows the same pattern as other fat tree like topologies, such as butterfly networks. The downwards routing is generally referred to as destination tag routing, simply uses the port labels as shown in figure 4.3.

Figure 4.3. Flat addressing packet format

4.3.1.2. Source-destination Addressing

XGFT is network architecture for parallel processing as already explained in literature review. The addressing used for this architecture according to (Kariniemi, 2006) is based on the space encoding used by TB (Turn Back) and TBWP (Turn back when possible) schemes. The addresses are of integer vectors which specify the routing path, both the source address and the destination address are attached into the packet header to ensure correct shortest path calculations (Kariniemi, 2006). We will show that our proposed bit sliced routing algorithm performs better. In XGFT two routing options exist, the up routing and the down routing. In the up routing, a packet is routed in the upwards direction until it finds the common routing switch ancestor or reaches the root switch of the destination node. The common ancestor switch can be found, by comparing the destination address with the source address carried along with the message at each switch stage of the network. Once the ancestor switch is reached, the down routing checks the destination address to determine the proper port to route the packet through to reach its destination.

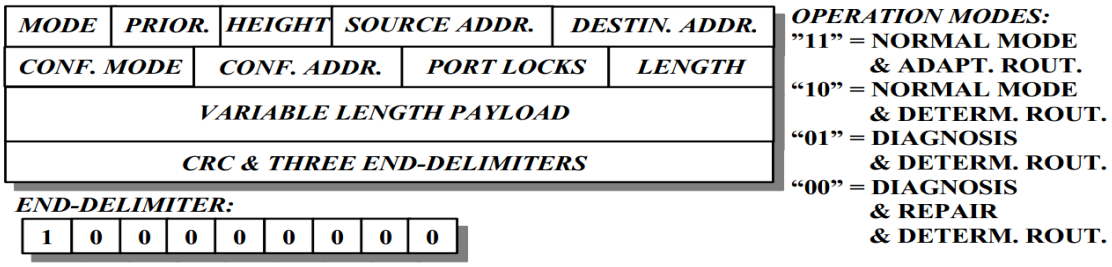


Figure 4.4. Packet structure (Kariniemi et al., 2006)

According to figure 4.4, the main addressing carried along with the packet is in the format of source address, destination address and the data PAYLOAD as illustrated in the following simplified diagram on figure 4.5.

Figure 4.5. XGFT addressing packet format.

4.3.1.3. Destination addressing

Destination addressing is mainly used in K-ary n-tree topologies. K-ary n-tree topology as explained in the literature review is a popular architecture used in parallel computing that is a rooted tree where each node consists of k-children and a constant arity switches (Pertini, 1998). K-ary n-tree uses minimal routing between pair of nodes by sending a message to one of the nearest common ancestors of both source and destination and from there to the destination. Each message experiences two phases an ascending phase to get to a nearest common ancestor, followed by a descending phase. The addressing utilised in k-ary n-tree consists of the Destination addresses being carried along in the header flit, but unlike XGFT it does not carry the source address. The destination address is compared to the address of the switches to find the common ancestor switch.

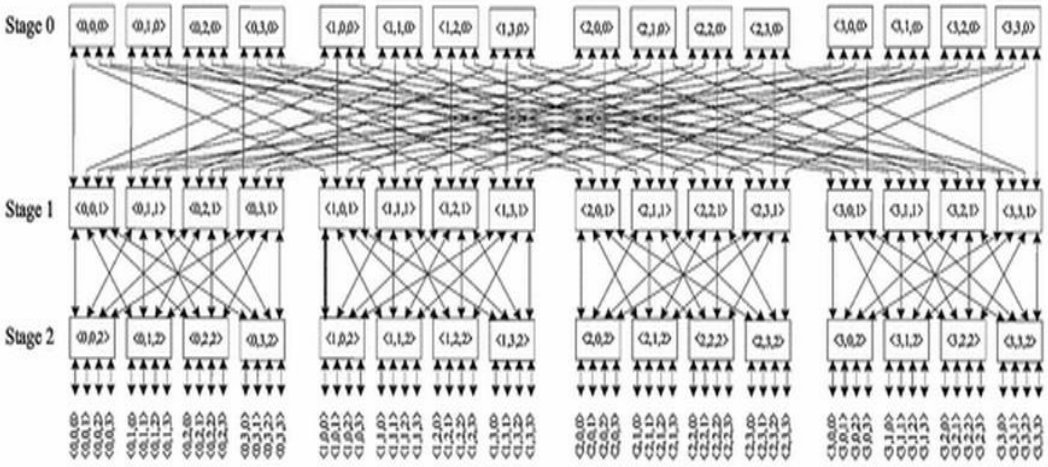


Figure 4.6. A 4-ary 3-tree (Petrini, 1998)

The destination which is embedded into the message header is used to route it to its destination and the bits are used to select the output port. The packet format can be illustrated in figure 4.7.

Figure 4.7. K-ary addressing packet format

4.3.1.4. Sliced addressing routing

We propose slice addressing as the addressing scheme for the Znode. Sliced is a source routing algorithm that determines the common ancestor level at the source and carry it along in stream of bits. Sliced addressing is selected as the best procedure in simplifying the routing of the messages, and minimising the latency. Its simplistic features along with the feasibility to scale, makes it an optimal routing solution especially for large number of processors compared to other addressing schemes used in fat tree class topologies.

A message that is travelling up to level i, before starting its downward direction would have carried the routing overhead of ($i + \log_2 m$) where i is the level and m the number of Znodes. Like any other routing of the fat tree family, in the downwards direction the message will be stripped off its upward routing overhead, remaining with ($\sum_{j=1}^{i} \log_2 z_j$) bits.

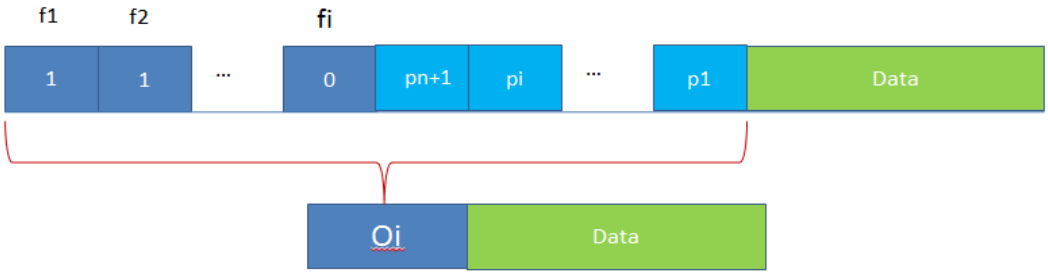


Figure 4.8. Illustrating sliced addressing used in Znode tree architecture

The implementation version of the definition 4 can be illustrated in equation 4.1. This eequation allows us to utilized sliced routing as:

$$f_i = \begin{cases} 1 & if\ (p_n, p_{n-1} \dots p_{i+1}) \neq (q_n, q_{n-1} \dots q_{i+1}) \\ 0 & otherwise \end{cases} \quad (4\text{-}1)$$

When fi=0, the level i holds the common ancestor routing switch. Equation 4.1 can be implemented in hardware at the source processor. The routing switches require no addressing hardware.

4.3.2. Latency incurred by routing and address overhead

The latency for each addressing mentioned earlier (flat, source-destination, destination and sliced) depends on the configuration of the system such as the number of levels and the number of Znodes in the Snode. It can be generally determined with the use of the following generic diagram (figure 4.9), along with the values for each latency variables are illustrated in the table 4.1 below. The latency variables are subject to the number of Znodes, the addressing used and whether the transmission is internal or external which will be explained further later.

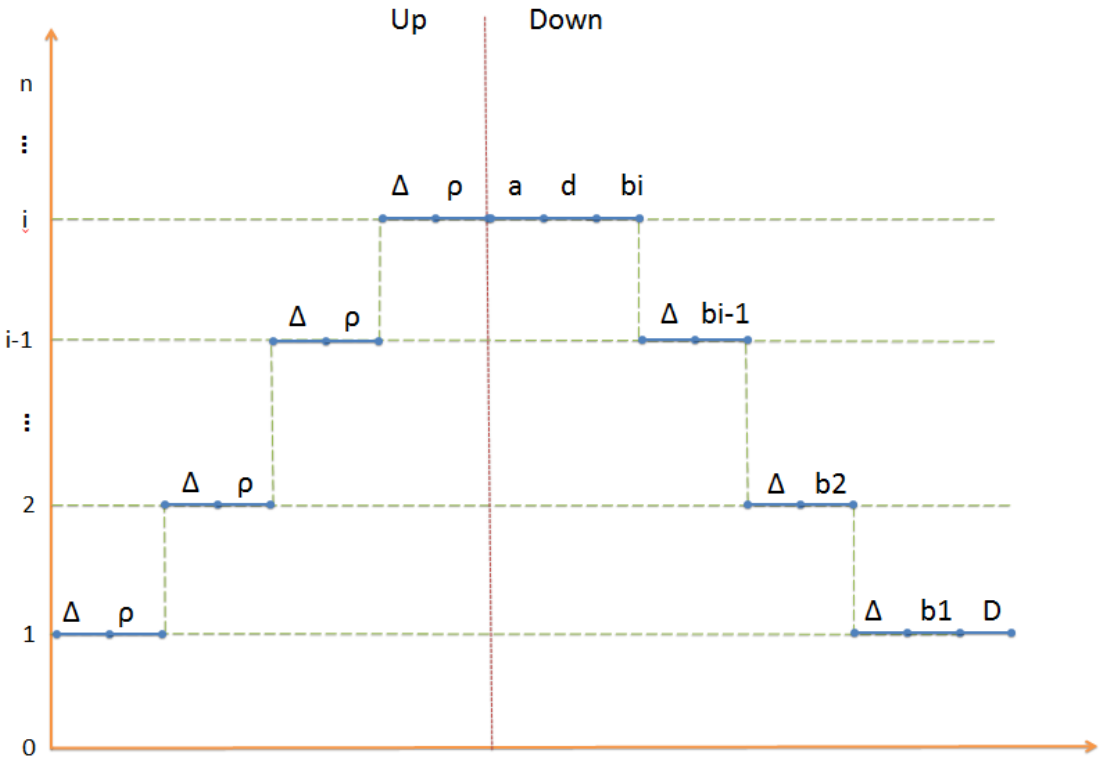


Figure 4.9. General latency diagram

Δ is the propagation delay between any two switches at level i, D is the data payload of the packet, (ρ) is the routing information bit to find the common ancestor, a is the overhead address to determine the Znode within the Snode which is equal to $\log_2 m$. (d) is defined as the extra propagation delay that is incurred when the message is forwarded to the next Znode as inter-Znode traffic. Intra-Znode traffic the extra propagation delay (d) is zero. (bi) is the address overhead used to determine the next zone that the packet will be forwarded to in the downwards direction, it is simply the port labels of the routing switches.

From figure 4.9 the latency can be deduced, in an empty network without any traffic as

$$T_i = (2i) * \Delta + (i * \rho) + a + d + \sum_{j=1}^{i} b_j + D \qquad (4\text{-}2)$$

Where $(2 * i) * \Delta$ is the propagation delay, $(i * \rho) + a + d$ is the routing overhead $\sum_{j=1}^{i} b_j$ is the destination address, and D is the data payload. In a more realistic scenario with a non-empty network the packet would have also encountered a queuing delay (Q) which would have been added to the overall propagation delay. The queuing delay is added in the simulation, which is explained in the next chapter, based on the distribution of the traffic.

	M=1				**M>1(internal)**				**M>1 (external)**			
	ρ	**a**	**d**	**bi**	ρ	**a**	**d**	**bi**	ρ	**a**	**d**	**bi**
Flat	0	0	0	$\log_2 Zn$	0	$\log_2 m$	0	$\log_2 Zn$	0	$\log_2 m$	Δ	$\log_2 Zn$
Sliced	1	0	0	$\log_2 Zi$	1	$\log_2 m$	0	$\log_2 Zi$	1	$\log_2 m$	Δ	$\log_2 Zi$
Source - destination	$2\log_2 p$	0	0	0	$2\log_2 p + 2\log_2 m$	0	0	0	$2\log_2 p + 2\log_2 m$	0	Δ	$\log_2 Zi$
Destination	$\log_2 p$	0	0	0	$\log_2 p + \log_2 m$	0	0	0	$\log_2 p + \log_2 m$	0	Δ	$\log_2 Zi$

Table 4.1. Parameters for each different routing in the Znode

4.3.2.1. Flat addressing average latency

In flat addressing upward direction routing forwards the packet all the way up to the top level root switch. The average latency in flat addressing can be calculated as:

$$T_{n\_F} = (2 * n) * \Delta + log_2 P + D \qquad (4\text{-}3)$$

Where P is the processors number, D is data payload, n is the level and Δ is the propagation delay through the links.

4.3.2.2. Source-destination addressing average latency

In the source-destination addressing, the message carries both source and destination addresses, which adds more overhead and hence higher latency. The (a) latency is equal to zero since it is already carried in the routing overheads. The average latency can be expressed as:

$$T_{n\_SD} = \Delta * (n+1) + (n+1)\, log_2\, P + \frac{1}{n} * \sum_{j=1}^{n-1} (n-j+i) b_j + D \qquad (4\text{-}4)$$

4.3.2.3. Destination addressing average latency

The destination addressing is similar to the source destination mentioned above with the difference that destination addressing does not carry the source information in the header. The average latency can be expressed as:

$$T_{n_D} = \Delta * (n+1) + \frac{1}{2} * (n+1) * log_2\, P + \frac{1}{n} * \sum_{j=1}^{n-1} (n-j+i) b_j + D \qquad (4\text{-}5)$$

4.3.2.4. Sliced addressing average latency

In sliced addressing the upward direction that the message will be routed to is based on the levels of the configuration which is determined by ($\Delta + i * \rho$). The average latency can be expressed as equation 4.7.

$$T_{n_S} = \Delta * (n+1) + \frac{1}{2} * (n+1) * \rho + D + \frac{1}{n} * \sum_{j=1}^{n} (n-j+i) b_j \qquad (4\text{-}6)$$

The general definition of the latency on more than one Znodes for both sliced and flat addressing can be expressed in the following equation:

$$T_{m>1} = T_n + \frac{\Delta}{2} + log_2\, m \qquad (4\text{-}7)$$

For source-destination and destination routing addressing, in the case of many Znodes we can write:

$$T_{m>1} = T_n + \frac{\Delta}{2} + \frac{log_2 p}{2 * n} + (n+1)\, log_2 m \tag{4-8}$$

For the proof regarding the derivation of the latencies equations for each addressing see Appendix A.

To identify the most appropriate addressing, the latency was calculated with the equations mentioned earlier without including the data payload. One can find the addressing latency with the data payload by adding 32 bits to each latency, as the most commonly payload is 32. As illustrated in the results obtained in figure 4.10, both source-destination and destination latencies increase proportionally as the level increases due to the fact that they carry a heavier header. On the other hand, flat and sliced addressing illustrates promising results with a lot lower latency. Although flat addressing starts to slightly increase as the processors increase, thus in a higher number of processors with high traffic flat addressing will suffer, as illustrated in the performance analysis chapter.

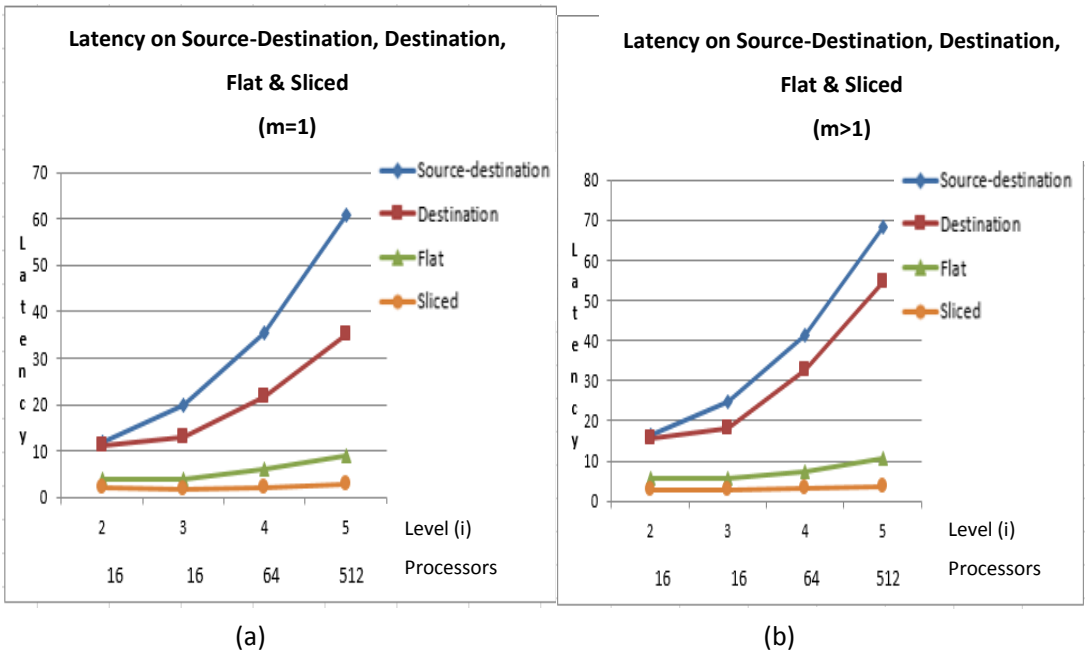


Figure 4.10. a) Illustrating a comparison of the addressing latencies on m=1, p=16-512 level=2-5 b) Addressing latencies on m=2, p=16-512 level=2-5

4.4. Optimal configuration

To find the optimal configuration among the space of all endless Znode topologies a simulator was developed that uses the power consumption as the objective function and applies traffic constrains in an attempt to find the best topology configuration. The simulator takes the number of processors as variable and runs a set of constraints in order to find the optimum configuration of both zones and switches to achieve the higher performance with the lower cost.

4.4.1. Objective Function

The objective function can be declared as the minimisation of the power consumption that is proportional to the cost of the architecture. By minimizing the cost of the architecture, one can minimise the power associated with it. The connectivity cost for each level depends on the number of ports, the number of zones, and the number of routing switches. Overall the cost of the Znode of level n can be expressed as:

$$Cost = \sum_{i=1}^{n} R_i^T * L_i^2 \qquad (4\text{-}9)$$

Where R_i^T is the total number of routing switches per level, and L_i is the total number of links per routing switch at level i. For the equation of L_i whether backwards or forward connectivity refer to Chapter 3 section 3.4.4. The total number of Routing switches and ports per level i, which determines the complexity of the level and hence the total complexity of the network is deduced in Chapter 3 section 3.3 equation 3.2.

The relative power in dB can also be used as an objective function, which is defined as the decimal log of the ratio of the complexity cost of the Znode over the complexity cost of a single bar switch.

$$dB_{relative power\ gain} = 10Log_{10}\frac{Cost}{(\prod_{i=1}^{n} Z_i)^2} \qquad (4\text{-}10)$$

4.4.2. Constraints

The Cost objective in equation 4.10 is subject to a several sets of constraints that ensure a high performance for the topology. This is achieved by setting the number of routing switches, ports per switch and number of levels to an adequate number that guarantees an overall minimum latency. The total number of processors is itself considered as a constraint limited by the number of zones, as defined in chapter 3 section 3.3, equation 3.1.

The number of processors is also subjected to other constraint $\leq m^2 \times n$. This constraint role is to ensure that there are enough links to carry side traffics within the Snode – intra-super node traffic- at each level n. When m=1, single Znode, this constraint is not used.

A link limitation constrain that limits the number of ports per switch to a specific threshold based on the current technology is also considered. In order to ensure that the number of links per routing switch does not go beyond a maximum threshold, we can write $L_i \leq \theta$ where θ the threshold that is imposed by the technology.

4.4.2.1. Constraints Analysis for forward (positive) connectivity

The connectivity constrains can be illustrated in the two following equations. The ratio between routing switches of different levels has to be a positive integer, as it defines the number of links per routing switch $G_i^+ = \frac{R_{i+1}}{R_i} \in Z^+$. The number of the routing switches per zone in each level has to increase from leaf to root for forward connectivity in order to satisfy the connectivity requirements of a fat tree, $R_i \leq R_{i+1} \leq R_{i+2} \leq \cdots R_n$.

The number of zones per level is the most important constraint. The zones and the routing switches are related to make sure that the numbers of ports per level are adequate enough to fully connect the number of sub-trees per level and hence minimise the delay in the network. The traffic constraint equation is based on queuing theory; the full proof is developed in appendix B, and can be simplified into:

$$Z_i \leq \frac{R_{i+1} \times \psi_{i+1}}{\prod_{j=i}^{n} Z_j} \qquad (4\text{-}11)$$

4.4.2.2. Constraints Analysis for Backward (negative) connectivity

The backward connectivity constraints follow the same logic as the positive connectivity constrains, but in the opposite order $G_i^- = \frac{R_i}{R_{i+1}} \in Z^+$. The number of the routing switches per sub-tree in each level has to increase from root to leaf $R_i \geq R_{i+1} \geq R_{i+2} \geq \cdots \geq R_n$. The traffic constraint for the backward connectivity, see appendix B, can be expressed as

$$Z_i \leq \frac{R_i \times \psi_{i+1}}{\prod_{j=i}^{n} Z_j} \qquad (4\text{-}12)$$

The full connectivity has no constraints on the number of routing switches, apart from that all other constraints apply. An example of a non-optimal configuration against an optimal configuration is illustrated in figure 4.11. Both the configurations include the same number of processors but their zones and routing elements differ. The optimal configuration is simulated and tested in terms of performance against non-optimal configurations in Chapter 5.

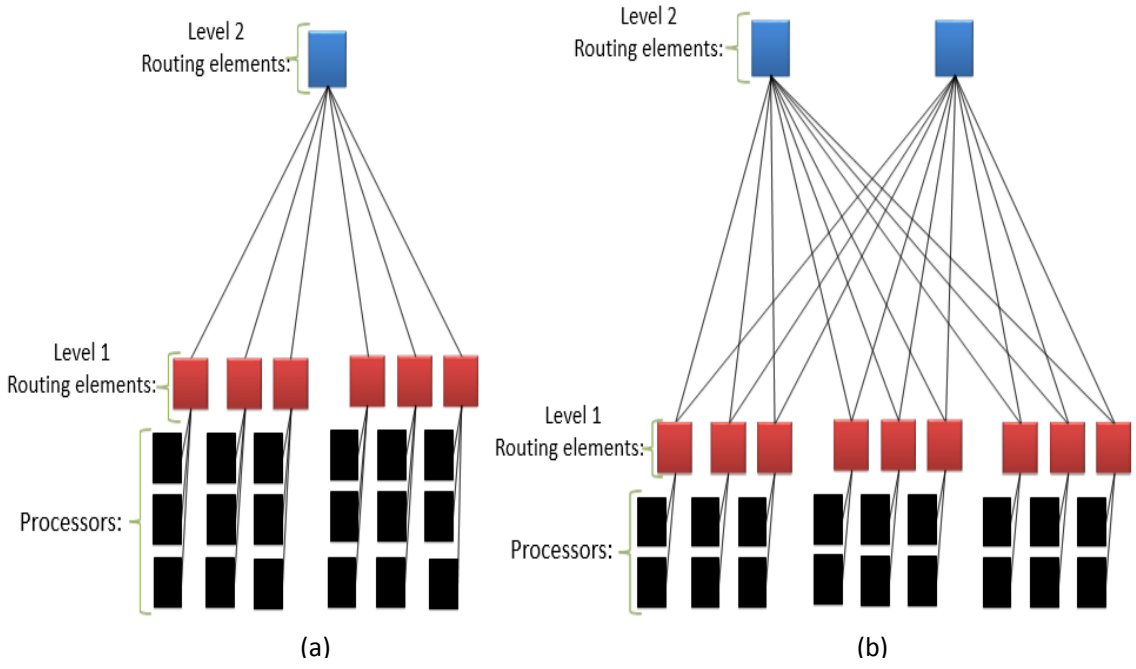


Figure 4.11. a) Non optimal configuration with p=18, b) optimal configuration with p=18

4.5. Summary

This chapter has provided an explanation for the proposed sliced algorithm a simple routing algorithm, sliced addressing for both Znode and Snode, whose routing information are generated at the source hence dismissing the address decoding per routing switch. Routing addressing modes including source-destination, destination, flat addressing and sliced addressing were analysed based on their latency. A comparison of those addressing schemes shows that the sliced addressing exhibits the lowest latency. Another contribution not identified in previous works is the identification the optimal Znode. The extraction of an optimal Znode from the large class of fat tree is based on minimising the overall cost network while maintaining lower message delay.

The next chapter will present the performance analysis of Znode and Snode and show how they behave under different network parameters and application patterns.

CHAPTER 5. PERFORMANCE ANALYSIS

5.1. Introduction

As discussed in Chapter 3, Znode architecture is a generalised fat-tree based topology, with additional features that promote its performance gain and infrastructure. Having already studied and presented Znode architecture in detail, we need to evaluate it by presenting some numerical and simulation results that demonstrate the applicability of our proposed architecture. Experimental simulations are performed both for Znode and Snode architectures. We also compare the results of Znode's delay and throughput against k-ary n-tree and XGFT configurations.

For the purposes of this research a parallel processing simulation was implemented in C++ called GalaNet, enabling us to simulate multiple Zoned nodes with multiple groups of levels along with the option of adjusting the properties of the channel links (appendix C). As explained in Chapter 3, the objective function simulator will also be used to extract the optimal configuration in terms of complexity – power consumption- and performance. Initially, the optimising simulator extracts the expedient parameters of the Znode that minimise the relative power, and hence the complexity of network and reduce traffic delay. Traffic delay is analysed for Poisson distribution with randomly generated traffic as shown in appendix C. To the best of our knowledge, other topologies do not address the optimisation of fat-tree class, but instead equate the number of input ports against the number of output ports. This is not always efficient because the traffic on the inputs ports can exceed the transmission capacity of the output ports.

5.2. Simulation

GalaNet simulator is an object oriented parallel event simulator that models the network architectural parameters of a target system (Refer to appendix C for the class diagram). It can successfully identify network performance bottlenecks such as high latency, large delay in queues and imbalances in the inter-communication of the target network.

GalaNet can effectively simulate variants and very large fat-tree class architectures. The simulator allows to analyse the performance of a target system by specifying a set of parameters. These can be the configuration of fat tree topologies levels, the number of zones (sub-trees) along with the routing switches per level and the number of Znodes in the super node configuration. The connectivity is adjusted by setting the connectivity degree of the switches, the connectivity between the levels and the connectivity between the layers along with number of up, down and side links. Once the configuration of the system is specified, the simulation environment is set by adjusting the message size, the message inter-arrival time, the payload distribution, the routing, the addressing mode and the switching operation of the routing elements. Generally, the simulation results are centered around the message delay (nanoseconds) and throughput (Gbits/sec). A copy of the parameters file can be viewed in Appendix D. A network map file and a statistics file are generated after each execution to view the connections between the switches and the processors and trace all messages sent and received by each processor, refer to appendix E for an example of the network map file.

5.2.1. Traffic patterns

Since the performance is greatly influenced by the application patterns, for our simulation we have adopted five widely used synthetic traffic patterns namely random, round-robin, bit-reversal, bit-complement, transpose and hotspot. In random pattern applications, a source processor selects a random destination and each processor has an equal probability of receiving messages. Random pattern is the most common static pattern widely used to evaluate parallel interconnections (Dally, 2004). Round robin chooses the destination based on a circular order; the traffic in round robin is distributed equally, one processor transmits to the next processor in address cycle, making it simple and effective but not always realistic. In fat-tree networks, round robin performs better, because messages tend to remain mainly in the first level. In bit-reversal a fixed source and destination exist for each message generated. The bits in the address of the source processor are reversed to generate the destination address of a processor (Feng, 2011). Some algorithms based on FFT use this kind of address mapping. In hotspot, traffic is not distributed equally; a group of nodes or a single node receives more traffic than other (Rahmani, 2009). The hotspot traffic occurs in farming types of parallel and server oriented applications, where the hotspot node represents a very busy server. In transpose, each node sends messages only

to a destination whose address is the transpose of the two halves of the address of the source node. Large data vectors and matrices use a lot of transpose algorithms. In complement traffic the destination address is determined by the inverse bits of the source node (Gratz, 2010). Finally, we proposed another traffic pattern that we call weighted-random pattern. This traffic pattern is dedicated to fat tree classes and uses probabilities as weights by forcing messages to reach a particular level before going to random destinations - refer to Appendix D for traffic probability setting as illustrated in the parameters of the simulator. In fact, this application pattern allows us to compare the optimal Znodes configurations for like-to-like and still behaves like a random pattern in a general sense.

5.3. Evaluation of Optimum level

Initially, it was attempted to optimise our proposed architecture by finding an optimum number of levels that can be applied to all configurations of the Znode. However, our results show that an optimum level does not really exist, since it depends strongly on the number of processors and different levels offer the best result for various numbers of processors. As a general rule though, we found out that for a large number of processors a higher level provides better results and for lower number of processors lower levels will be more appropriate. This is not new as for the k-ary n-tree the number of level increases with the number of processors and decreases with the arity of the tree n= log (p)/log (k). In our case we expect the level to decrease with the size and number of the zones. Consequently, finding an optimal number of levels is not always trivial since it depends on the complexity of the routing elements and the number of processors. As a rule of thumb, higher level will be associated with higher processors.

5.3.1. Optimum level for 512 processors

In the following deduction, the number of processors is set to 512, and the optimum configuration of zones and routing switches for each level is based on the constraints discussed in the previous chapter. The best configuration for each level can be illustrated in table 5.1. In the case of level 3, two configurations with the same relative power consumption and cost have been generated by the objective simulator. The relative power consumption, as explained in chapter 3, is measured in terms of reduction of the network

complexity against the complexity of a single switch configuration with the same number of processors. The maximum number of links' threshold per routing element is set to 64 for all simulations. These are moderate switches which are available as off-shelf at reduced prices but of course higher ports switches are available too. Our analysis shows that if a larger number of ports are included into the calculations, we obtain the same results but with more options which eventually add to the cost and complexity of the routing switches. On the other hand, smaller number of ports per switch will require more levels to accommodate larger number of processors and hence increasing the latency of the Znode configuration. Therefore, a choice of a maximum of 64 links per switch is considered to be normal.

We have observed that levels 6 and 4 have the least complexity (-9.61dB). However, this does not guarantee they have the optimal performance in term of message delay and throughput, but within all possible solutions found at level 2 for example, the configuration $oZ_2 \overset{\text{def}}{=} \frac{(1,8)/(1,1)}{(8,64)}$ is the optimal.

Level	Zones(512 processors):						Routing elements						Links						Relative power
	1	2	3	4	5	6	1	2	3	4	5	6	1	2	3	4	5	6	
2	8	64					1	8					16	64					-7.27
3	8	4	16				1	8	32				16	8	4				-9.03
3	4	4	32				1	4	16				8	8	32				-9.03
4	4	4	4	8			1	4	16	64			8	8	8	8			-9.61
5	2	2	2	2	32		1	2	4	8	16		4	4	4	4	32		-9.03
6	2	2	2	2	2	16	1	2	4	8	16	32	4	4	4	4	4	16	-9.61

Table 5.1. Parameters of Optimum configuration for levels 1-6 with 512 processors

The performance evaluation of all the optimum Znodes at all levels 2 to 6 is carried out by simulating all network offered loads with the parameters 1Gbit/sec channel rate, random traffic pattern and uniform payload distribution (refer to Appendix D for the parameters). We express the offered load as the ratio of the payload normalised by the channel rate (1Gbit/sec) to the average Inter-arrival time of the exponential distribution.

For example, an offered load of 100% would require a message length of 32bits, and inter-arrival time of 32 nanoseconds. The optimum level for 512 processors is level 2 and offers the lowest message delay, along with the highest throughput across all offered loads, see figures 5.1 and 5.2, at the expense of more links per switch, 16 links in the first level and 64 links in the second level, see table 5.1. Figure 5.3 shows the detailed message delay around the offered load of 20% in terms of random and weighted-random traffic pattern. It is illustrated that level 2 exhibits the lowest message delay for both application patterns.

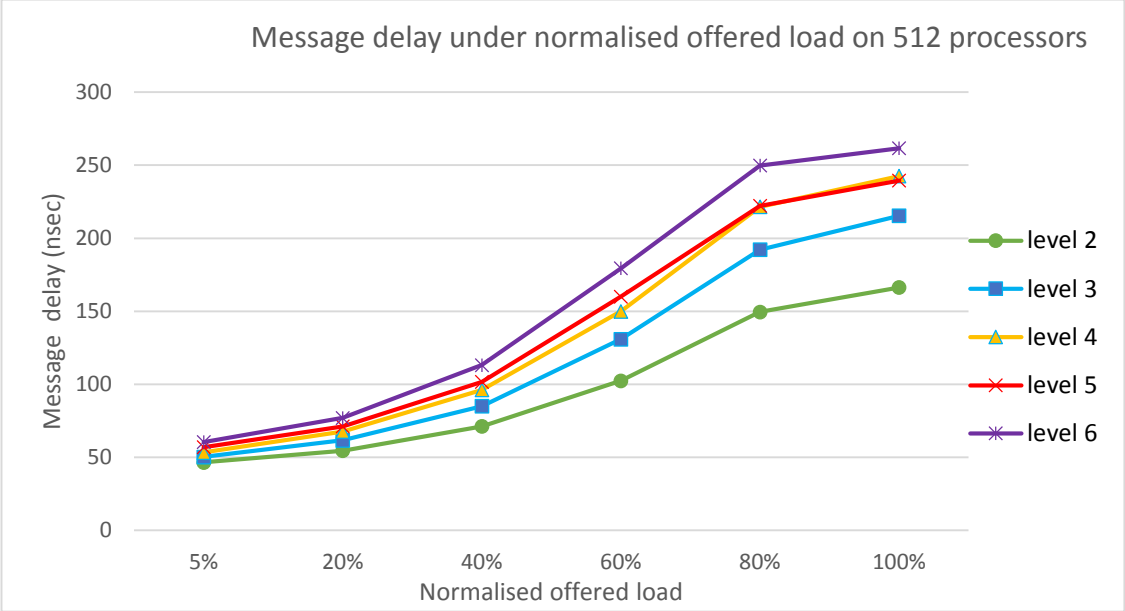


Figure 5.1.Message delay on 512 processors from levels 2 to 6

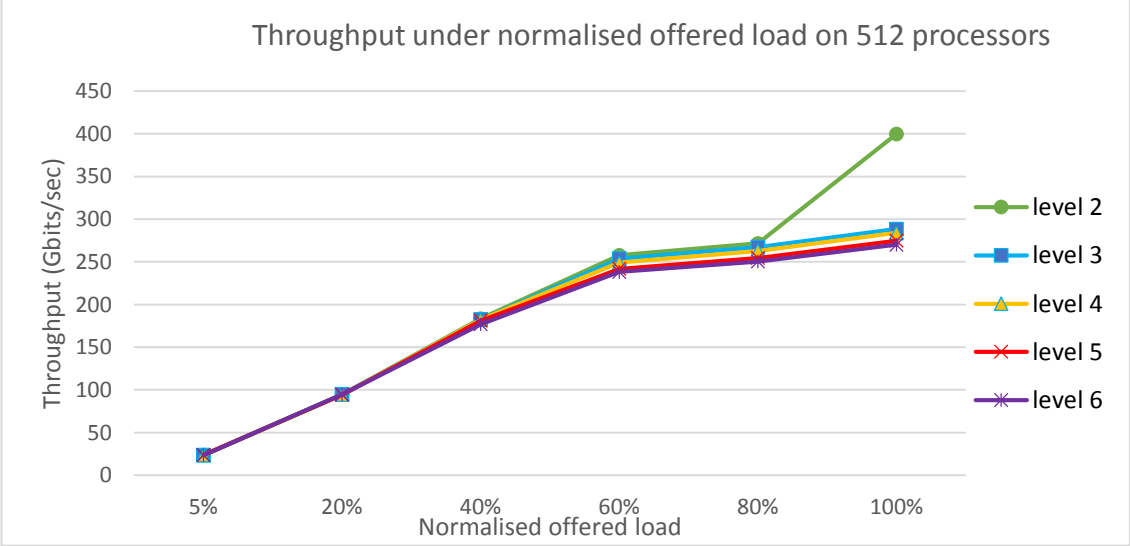


Figure 5.2. Throughput on 512 processors from levels 2 to 6

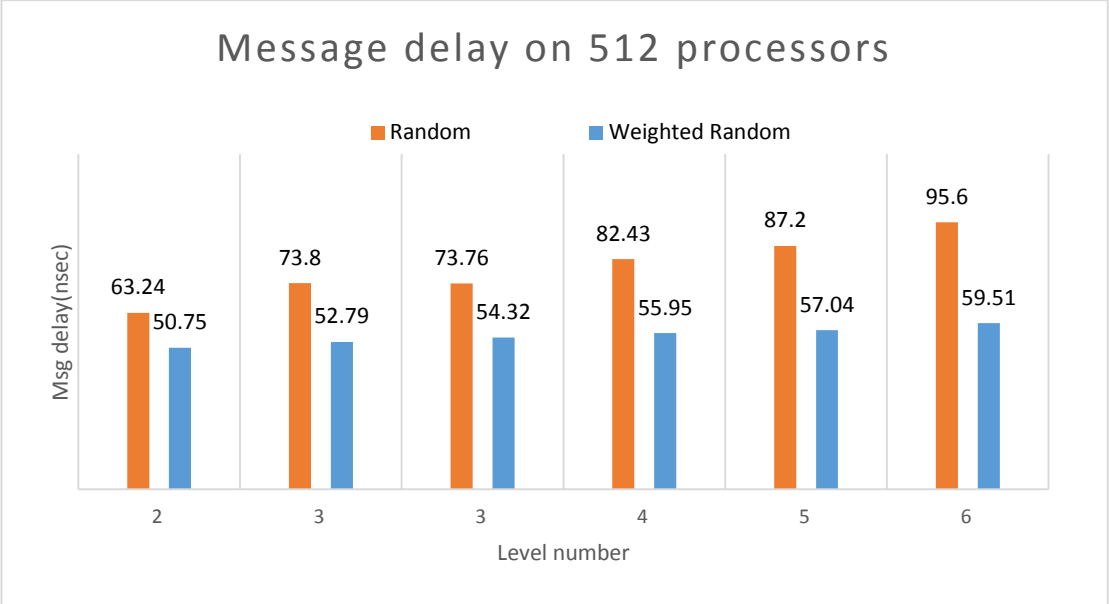


Figure 5.3. Message delay at load of 20% on 512 processors from levels 2 to 6

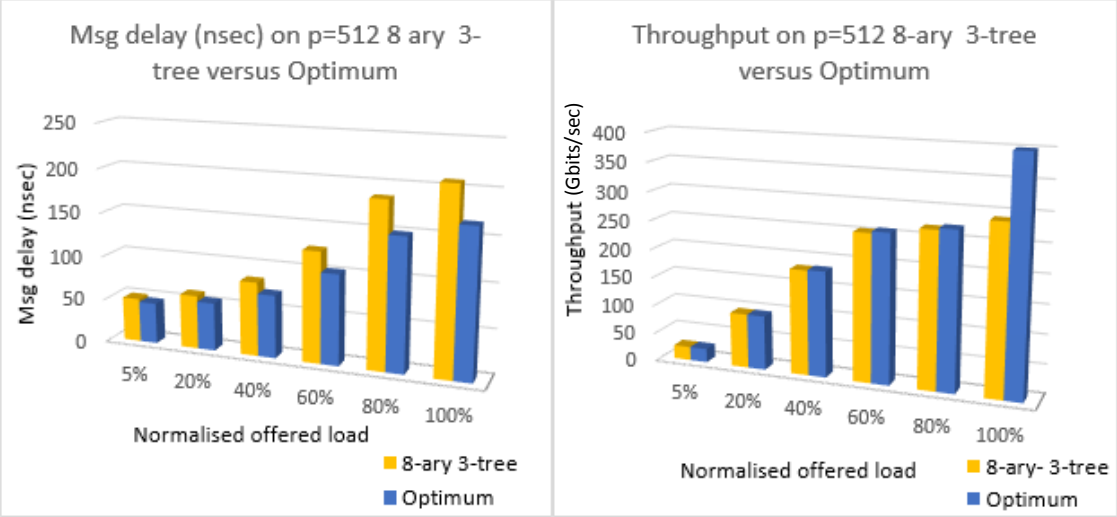


Figure 5.4. Message delay Optimum versus 8-ary 3-tree

The k-ary n-tree was not selected by the objective simulator as being the optimal fat tree configuration since the deduced Znode configuration has lower delay and better throughput, especially at higher loads. As shown in figure 5.4 once both 8-ary 3-tree and optimum configuration for 512 processors were tested under Destination addressing which is the one adopted in k-ary n-tree it was found out that the optimum configuration

performs significantly better under all offered loads. This result proves that our optimisation model can successfully optimize fat tree variants.

5.3.2. Optimum level for 1024 processors

In the case of 1024 processors, the optimum Znodes configurations for levels 2 to 6 are shown in table 5.2, along with the number of links per routing switch at each level and the relative power consumption.

Level	Zones (1024 processors):						Routing elements						Links						Relative power
	1	2	3	4	5	6	1	2	3	4	5	6	1	2	3	4	5	6	
2	32	32					4	16					36	32					-7.60
3	8	8	16				1	8	64				16	16	16				-11.07
4	2	2	4	64			1	2	4	16			4	4	8	64			-10.28
5	2	2	2	2	64		1	2	4	8	16		4	4	4	4	64		-10.60
6	4	2	2	4	2	8	1	4	8	16	64	128	8	4	4	8	4	8	-12.04

Table 5.2.Parameters of optimum configuration for levels 2-5 for 1024 processors

Table 5.2 indicates that configuration $oZ_6 \overset{\text{def}}{=} \frac{(1,4,8,16,64,128)/(1,1,1,1,1,1)}{(4,2,2,4,2,8)}$ has the lowest complexity cost, that is the lowest relative power of (-12.04dB). This is due to the lowest number of links per routing elements which is no more than 4 or 8. However, the results for message delay and throughput, figures 5.5 and 5.6, show that levels 3, 4, and 5 achieve the lowest delay and highest throughput across all the offered loads, with level 3 slightly offering the best performance. Level 2 is shooting up on figure 5.5 due to the fact that the traffic is queued as a higher level would have been required to support the traffic.

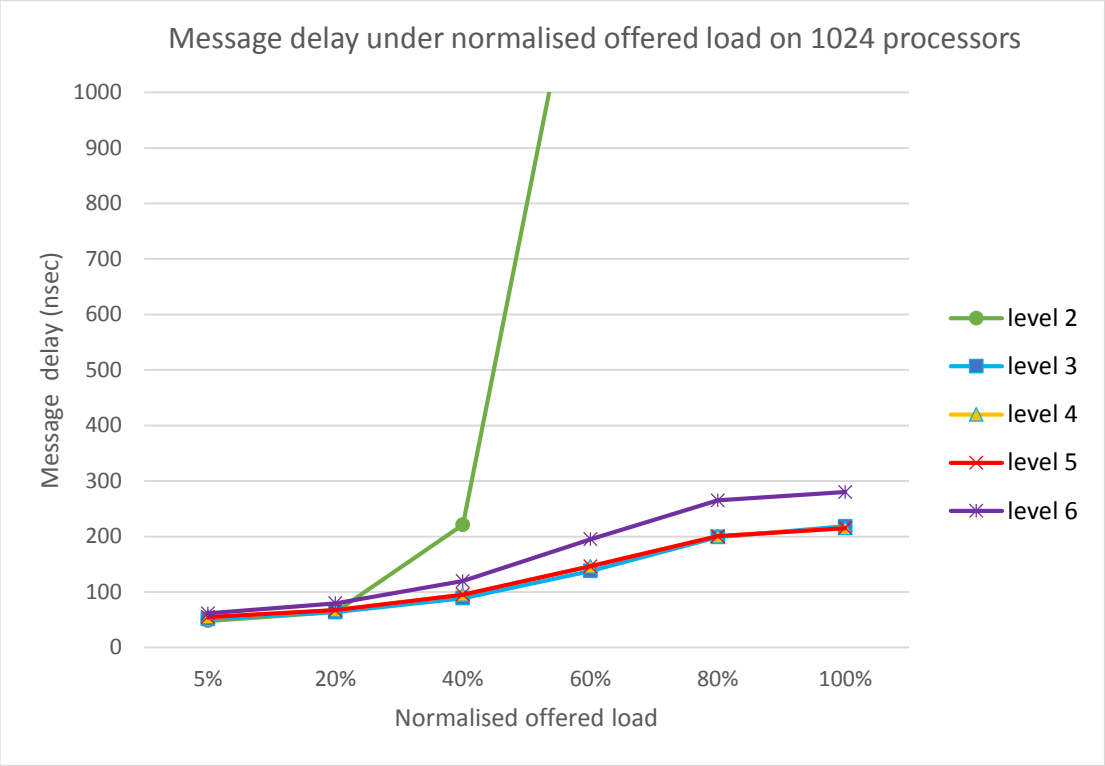


Figure 5.5. Message delay with 1024 processors for levels 2 to 6

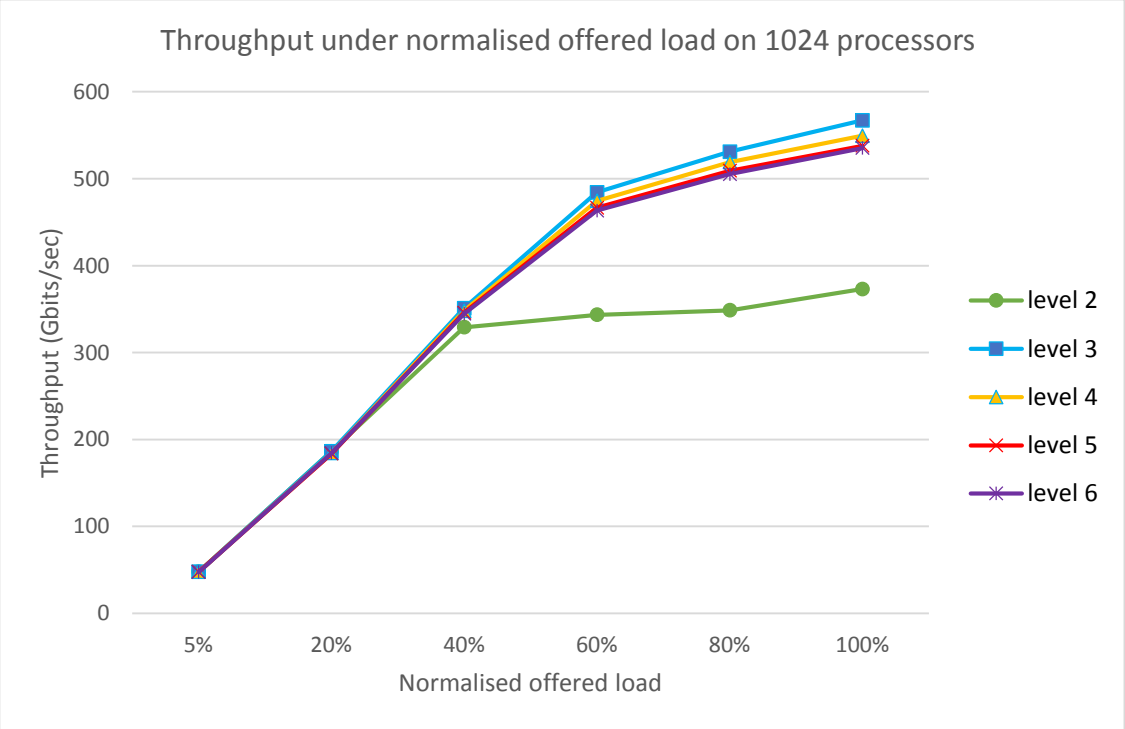


Figure 5.6. Throughput on 1024 processors for levels 2 to 6

5.3.3. Optimum level for 2048-8192 processors

For 2048 to 8192 processors level 4 is the optimal level and level 8 is the optimal level for 16384 processors. Larger the number of processors that are accommodated by the network, higher is the optimal level in term of performance and cost trade-off.

Figure 5.7 shows the message delay and throughput with varying offered load whilst increasing the number of processors from 512 to 4096. We notice that when the offered load increases, the message delay and the throughput increase accordingly. What is interesting are the not so high differences in the number of processors, which proves that Znode can scale efficiently with the number of processors. We further evaluate this point in the next sections. As expected, the throughput increases proportionally to the normalised load. This is due to the fact that while the load increases, so does the number of the transmitted messages and since throughput represents the average rate of successful messages delivered per unit of time the throughput increases as well.

Figure 5.7. Message delay on 512-4096 processors under normalised load

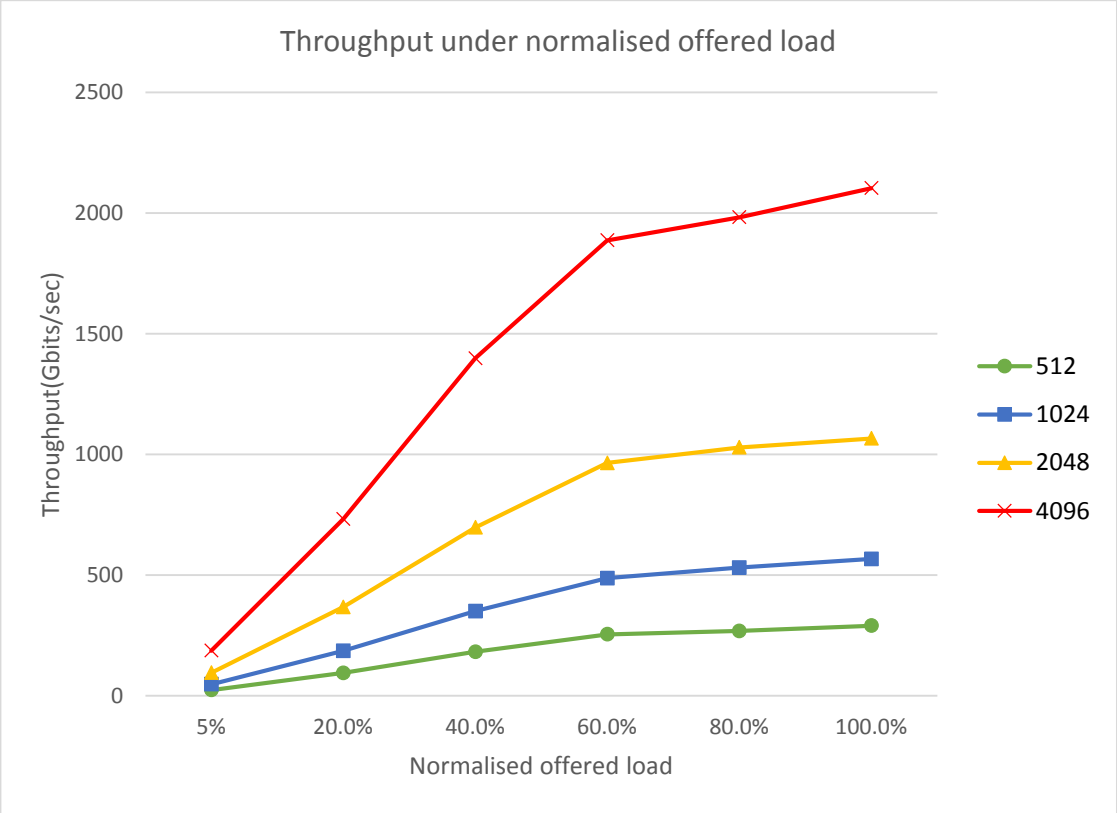


Figure 5.8. Throughput under normalised load on 512-4096 processors

5.3.4. Optimum configuration under XGFT architecture

In this section we find out that the optimal configuration of Znode can be applied to other fat tree variants also. The extended generalised fat tree "XGFT" explained in chapter 3 is used as the test subject, where two configurations of XGFT for 36 and 60 processors (Kariniemi, 2006) are evaluated against the optimum configuration for the same number of processors. The connectivity degree and functions of XGFT architecture, along with its source-destination addressing, have been utilised in the following simulation to provide a realistic comparison of both configurations under the XGFT environment. According to Kariniemi's example (Kariniemi, 2006) the XGFT configuration for 36 processors consists of 2 levels, the first level has 6 routing elements and the second level has 4, and for 60 processors, there are 3 levels each consisting of 15, 10 and 4 routing elements consecutively (fig. 5.9).

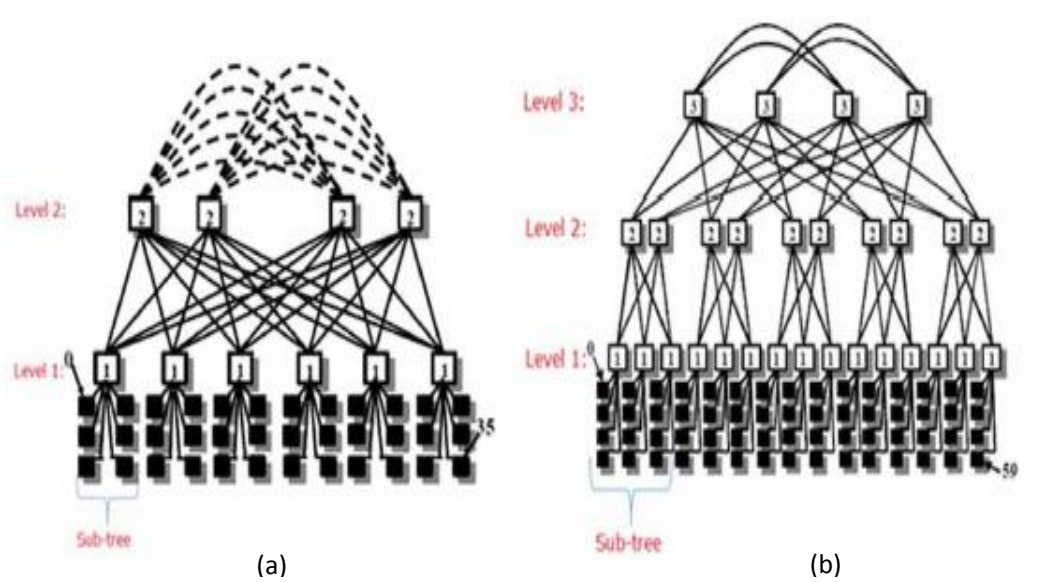


Figure 5.9. a) XGFT configuration on 36 processors b) XGFT configuration on 60 processors (Kariniemi, 2006)

The equivalent optimal Znode configurations for 36 and 60 processors are shown in table 5.3 and 5.4 and graphical representation can be illustrated in figure 5.10.

Level	Zones (36 processors)				Routing elements				Links				Relative power
	1	2	3	4	1	2	3	4	1	2	3	4	
2	3	12			1	3			6	12			-1.76
3	2	2	9		1	2	4		4	4	9		-1.58
4	2	2	3	3	1	2	4	12	4	4	6	3	-0.65
4	3	2	3	2	1	2	6	18	5	5	6	2	-0.70

Table 5.3. Parameters of optimum configuration for levels 2-4 for 36 processors

Level	Zones (60 processors)				Routing elements				links				Relative power
	1	2	3	4	1	2	3	4	1	2	3	4	
2	4	15			1	4			8	15			-2.87
3	3	2	10		1	3	6		6	4	10		-3.010
4	2	2	3	5	1	2	4	12	4	4	6	5	-2.60
4	2	2	5	3	1	2	4	12	4	4	8	3	-2.42

Table 5.4. Parameters of optimum configuration for levels 2-4 for 60 processors

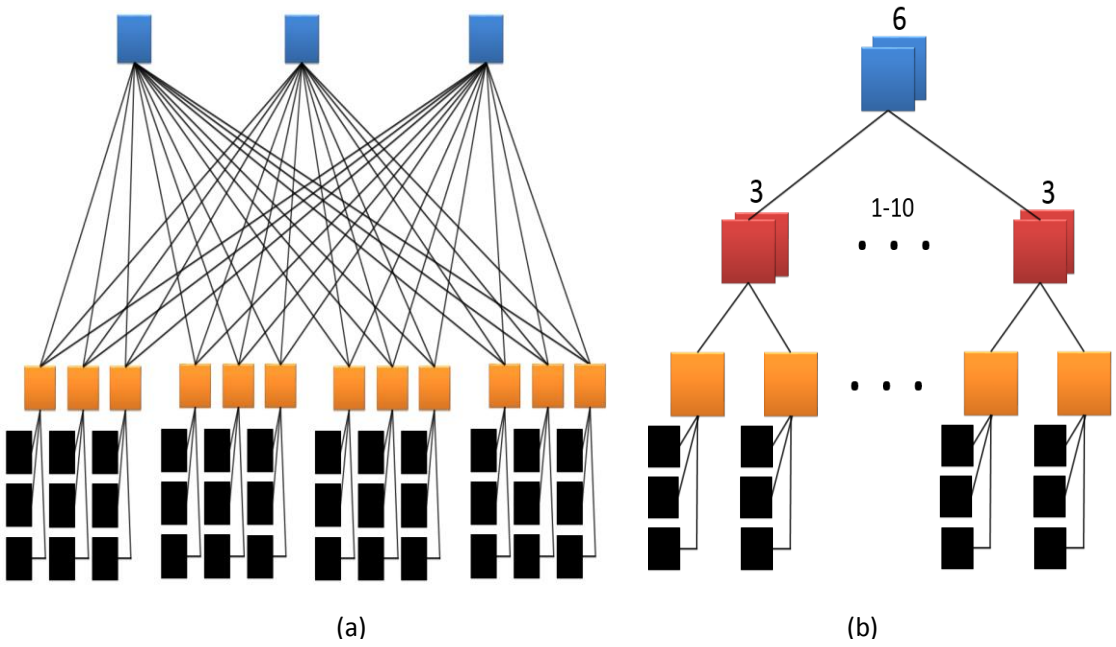


Figure 5.10. a) optimal configuration on 36 processors, b) optimal configuration on 60 processors

Figures 5.11 (a) and (b) show that level 2 gives the optimal Znode configuration in terms of message delay, throughput and relative power consumption (36 processors). For 36 processors the configuration consisted of 12 switches for the first level, with each switch connected to 3 processors, and 3 switches for the second level as shown in figure 5.10 (a). One of the optimal configurations for 60 processors is shown also in figure 5.10 (b) with 3 levels. Despite the fact that level 2 is the optimal configuration, we have selected level 3 to perform alike for comparing to XGFT (Peratikou & Adda, 2013).

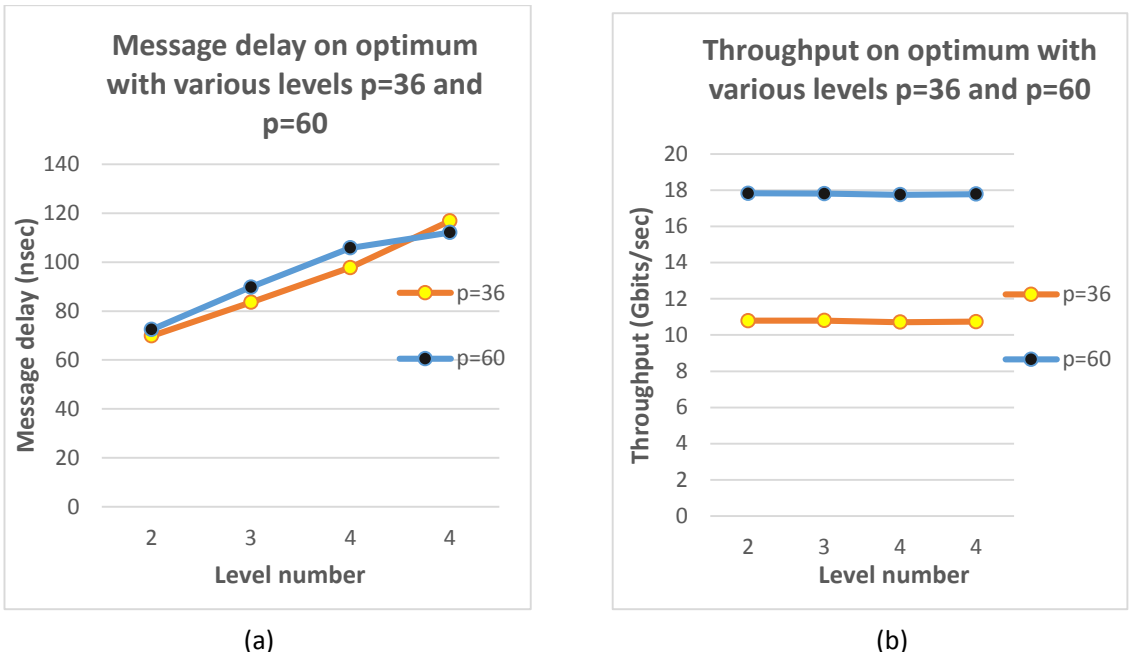


Figure 5.11. (a) Message delay (b) throughput for 36 and 60 processors under normalised load,

XGFT configuration and Znode optimal configuration on 36 processors were tested under various offered traffic loads all under the routing of the XGFT – source-destination routing. Both configurations performed similarly on a load of 5% to 50%, with the optimum configuration achieving slightly lower message delay of -1.00 to -2.00 nanoseconds compared to XGFT. When load increases to 60% the throughput difference between the two configurations are noticeable (Figure 5.13), and the message delay increases significantly for XGFT. This is due to the lack of ports interconnections to service the backed traffic at each level. If we zoom in the graph for loads higher than 70% (Figure 5.12) the difference in the message delay is more evident.

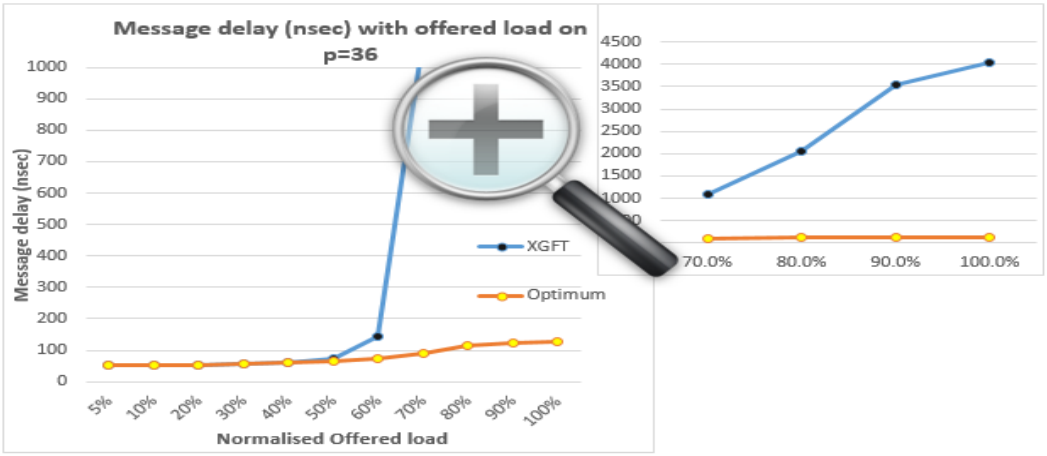


Figure 5.12. Message delay on various input rates for 36 processors

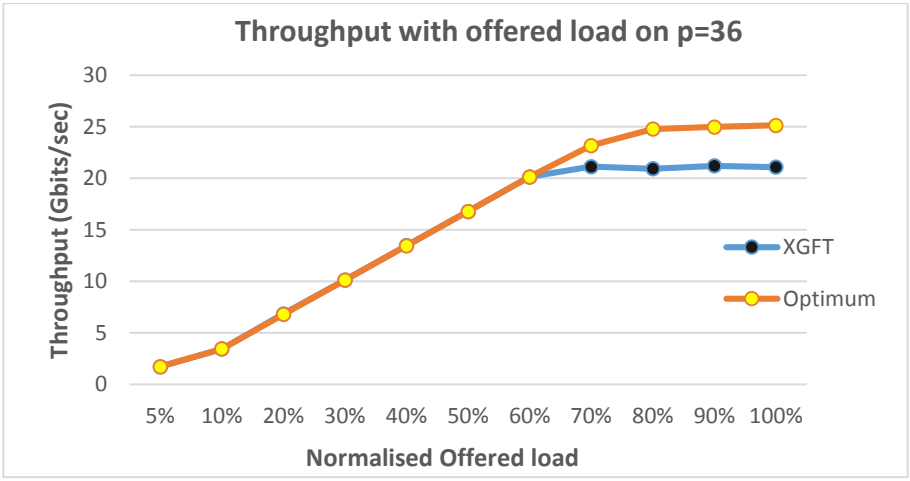


Figure 5.13. Throughputs under various input rates

Once the configurations were tested under an inter-arrival time of 45 nanoseconds, meaning 70% of the offered load, the difference between the two configurations becomes clearer. Figure 5.14 indicates that the message delay is lower for all traffic patterns in the optimum configuration and throughput is higher on the optimum configuration in all cases except on transpose where both configurations share the same value.

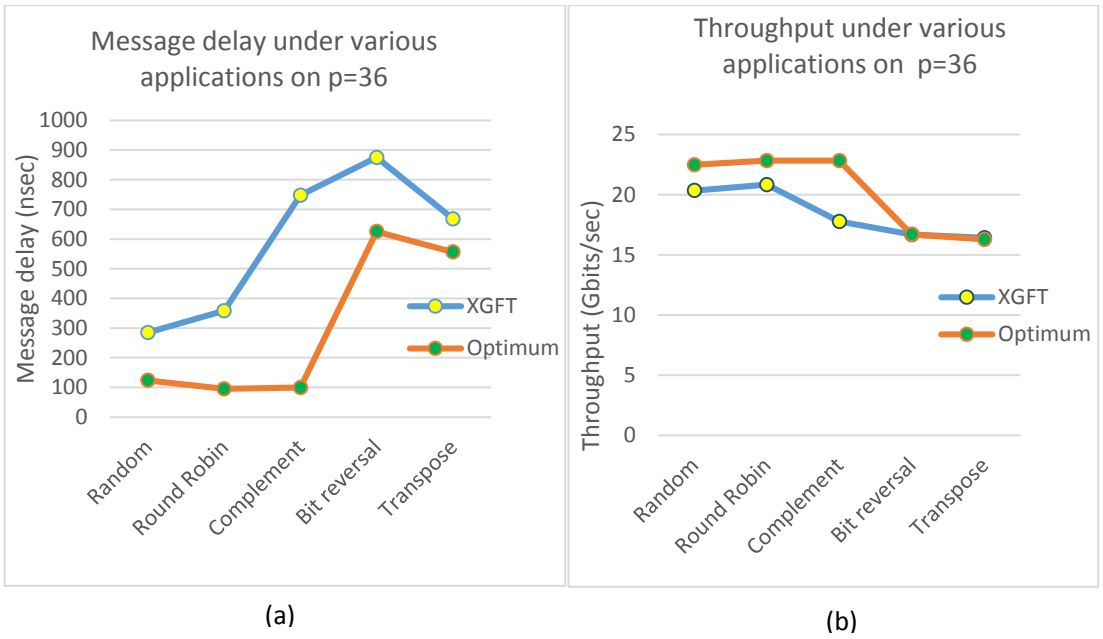


Figure 5.14. a) Message delay on 36 processors under different applications. b) Throughput on 60 processors under various applications.

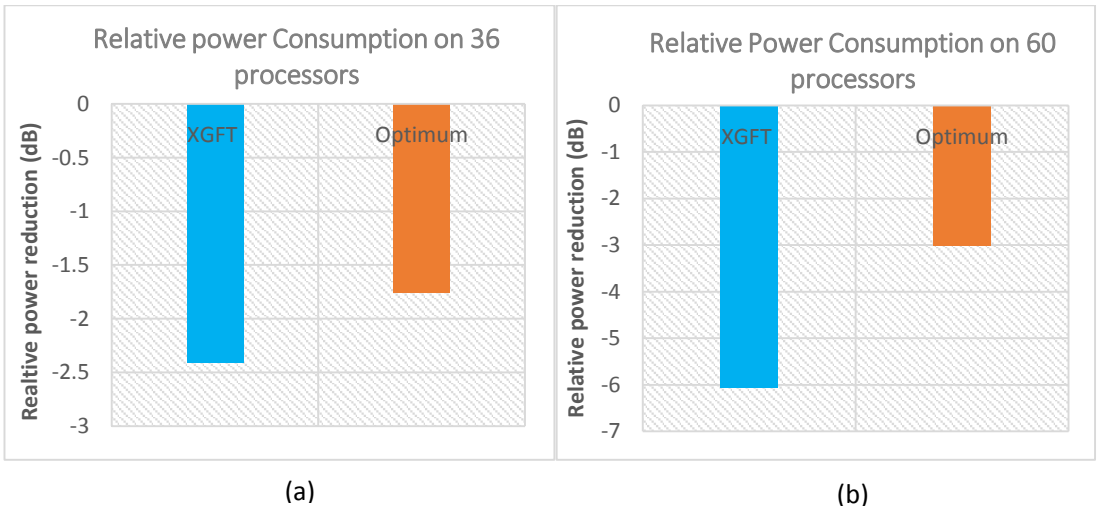


Figure 5.15. a) Relative power comparison of XGFT and optimum on 36 processors. b) Relative power comparison of XGFT and optimum on 60 processors

As indicated in figure 5.15 XGFT has lower relative power than the optimum Znode configuration. This is due to the extra links and number of routing elements that make up for a decrease in the message delay and increase in throughput. This is a trade-off between performance and power consumption (Peratikou & Adda, 2013).

The optimal XGFT for 60 processors consists of 20, 30 and 6 switches for levels 1, 2 and 3 respectively (figure 5.10(b)). Based on the simulation results illustrated in figure 5.16, it is identified that the optimum configuration performs significantly better on 60

processors than the non-optimised XGFT. The throughput is higher in the optimum configuration under most of the traffic patterns, with the exception of hotspot traffic in which both configuration performed similarly (figure 5.16 (b)). The message delay obtained in the optimum configuration follows a straight pattern with an increase towards transpose while in XGFT the message delay increases enormously under complement and transpose traffic .

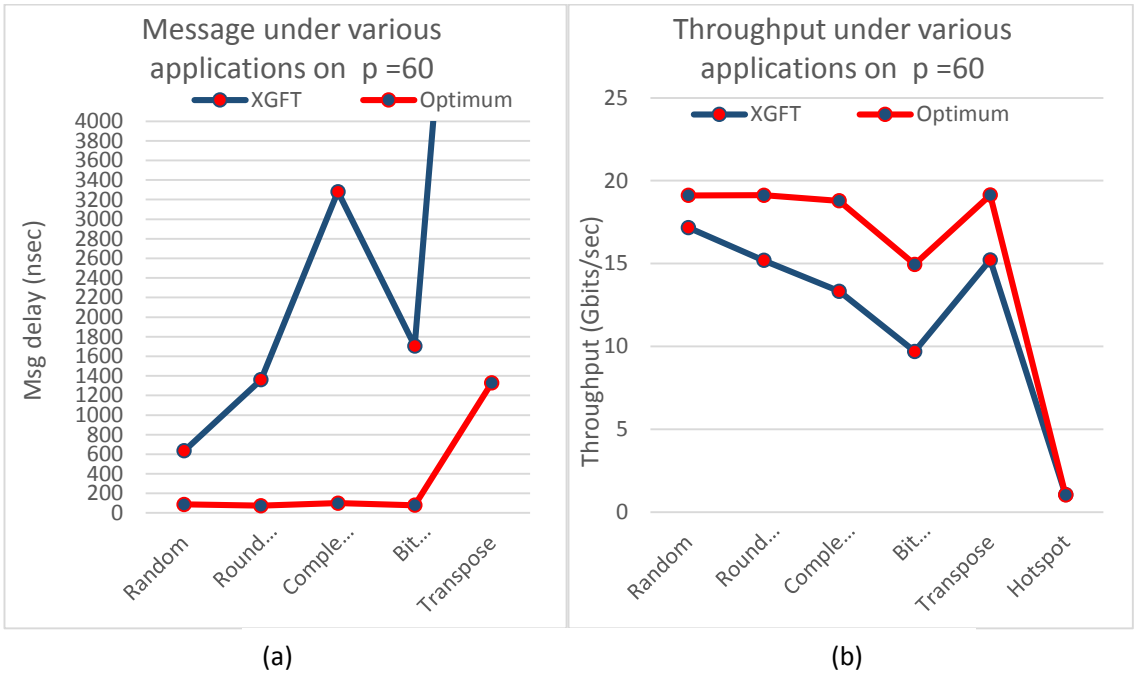


Figure 5.16. a) Message delay on 60 processors under various applications b) Throughput on 60 processor configuration under various applications

Throughout this section it has been clearly demonstrated that the optimum configuration can offer performance enhancements over the XGFT architecture. The results obtained are more than convincing that the optimum configuration can be widely used as a good technique for optimizing fat tree class topologies. The optimal configuration is used in all simulated scenarios for the remaining sections of this chapter.

5.4. Address routing

The sliced addressing explained in chapter 4 is proposed as an efficient routing address for Znode architecture. In this section we simulate the sliced addressing against the routing addresses used in other fat tree configurations such as source-destination

addressing, destination addressing and flat addressing. The simulations are performed while using the application patterns of random, round-robin, bit-complement, bit-reversal, and transpose. The configuration of the system is consisted of 1024 processors allocated to a single Znode with 3 levels consisting of 8, 8 and 16 zones respectively, with 1, 8 and 64 routing elements per zone, semantically represented as $Z_3 \stackrel{\text{def}}{=} \frac{(1,8,64)/(1,1)}{(8,8,16)}$. The simulation entails a fixed inter-arrival time of 100 nanoseconds and a fixed message length of 32bits, giving an offered load of 30%.

As shown in figure 5.17, the single-bit sliced addressing routing that we proposed has the lowest message delay, for all simulated traffic patterns. This is due to the single bit per hop (level) passed with the message rather than carrying along the entire source and the destination addresses. Flat addressing has the highest latency under bit–reversal. This delay is due to the waiting time in queues. We also observe the same issue under transpose traffic due to the fact that the message is forwarded all the way to the highest switching element. By the time it starts the downward direction, the communication overhead on its links increases the delay.

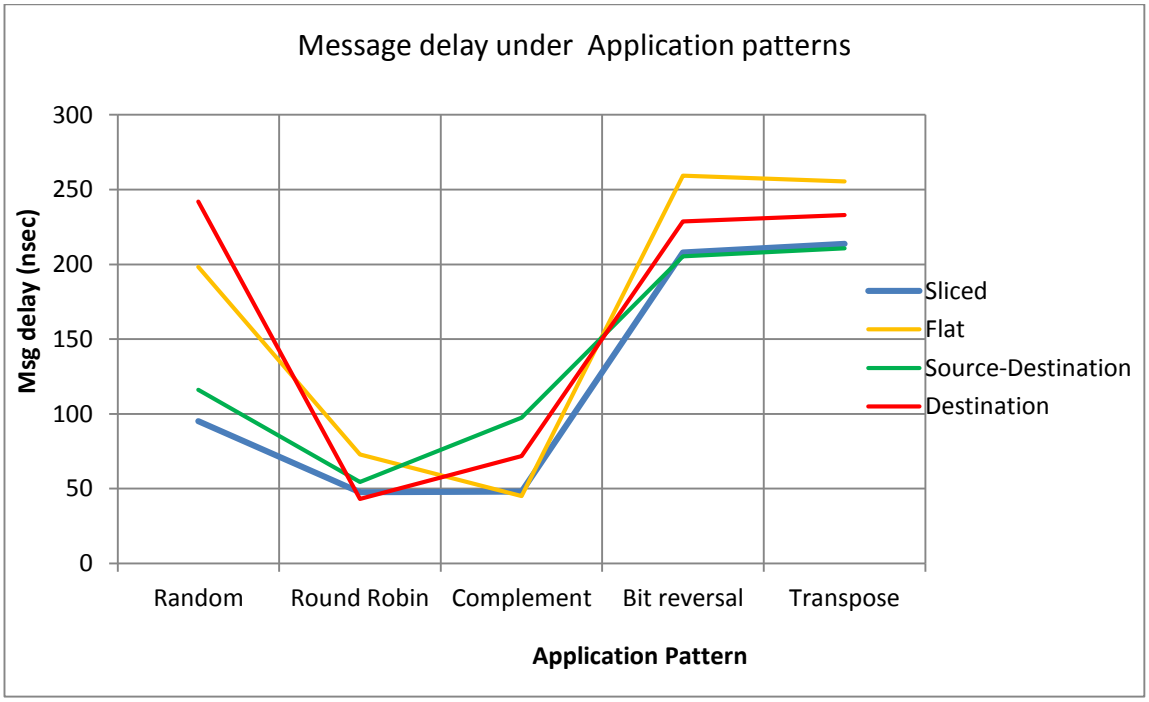


Figure 5.17. Message delay against addressing schemes under various applications.

5.5. Connectivity

The forward (positive) connectivity and the backward (negative) connectivity were evaluated with a configuration of 1024 processors, on level 3 with zones and rooting elements as illustrated on table 5.5. On the negative connectivity are used 8, 8 and 16 zones with routing elements of 128, 64, and 32. For positive connectivity, refer to table 5.2 section 5.3.2. Positive and Negative connectivity are evaluated on a single Znode, with inter-arrival time of 40 nanoseconds which is 80% of the offered load. We have chosen transpose for these scenarios as most interconnections suffer under transpose and bit reversal traffic pattern, refer to figure 5.17. The value for each connectivity degree can be illustrated in the following table along with the link threshold and relative power per connectivity. Generally, the negative connectivity consumes more power than the positive connectivity.

connectivity	Ψ values			links			Relative power
	1	2	3	1	2	3	
Positive	1	1	1	16	16	16	-11.07
	1	1	2	16	24	32	-7.85
	1	1	4	16	40	64	-3.22
Negative	1	1	1	9	17	32	1.98
	1	1	2	9	18	64	2.32
	1	2	2	10	34	64	56.36

Table 5.5. Parameters of optimum configuration for level 3 under various degrees of connectivity with a maximum threshold of 64 links per routing elements and 1024 processors

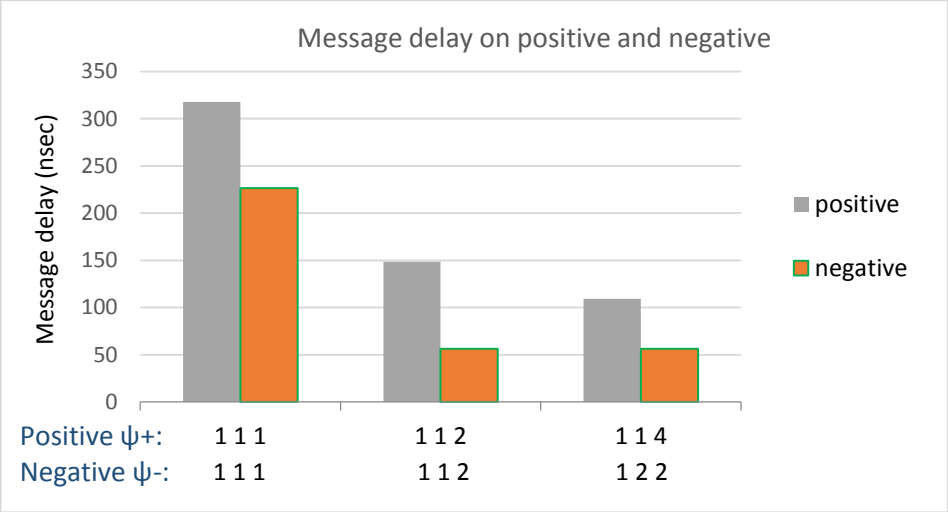


Figure 5.18. Message delay on positive and negative connectivity

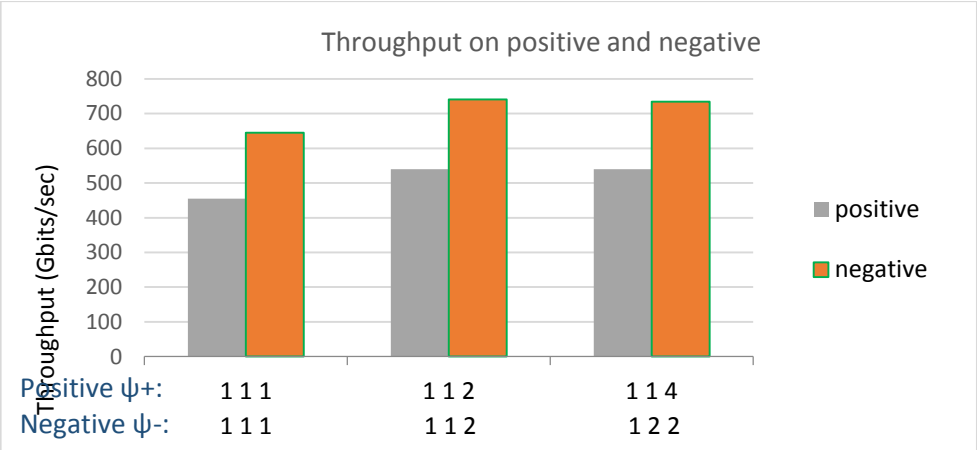


Figure 5.19. Throughput on positive and negative connectivity

From the figures 5.18-5.19 we notice a trade-off between power and performance. In both positive and negative connectivity, when the connectivity degree ψ is greater than 1, as more links exist between the levels, the message delay is reduced, at a price of slightly higher relative power consumption.

In figure 5.19 we observe a lower message delay in negative connectivity compared to positive connectivity. In negative connectivity the routing elements to the lower level are higher than the routing elements of a higher level allowing a faster transmission in downwards rooting where is needed the most. Also by increasing the routing elements on lower levels a higher number of available paths exist minimising the overall queuing and waiting time. To the best of our knowledge, no other fat tree has implemented the backward (negative) connectivity. The results show that negative connectivity with slight

increase in the connectivity degree gives better message delay at the expense of more power consumption.

5.6. Scalability

The capability of a network to scale and increase to accommodate larger number of processing power is important for large scale computing as it allows more processing elements to be added without major upheaval to its structure or performance. The network remains controlled and exhibits a stable grow (Martey, 2002). The most important factors that determine the scalability are the configuration of the routing elements, links and the flooding. Since Znode routing elements do not have to handle large amounts of information due to the lack of routing tables, the flooding is easily controlled. Znode can be scaled either by increasing the number of levels or by expanding the size of the super node, Snode, thus offering recursive scalability.

Fat tree class architectures are generally considered to be highly scalable (Infiniband-melanox, HPC-simula lab, 2012). According to Melanox, fat tree topologies can offer cost effective scalability. In fat trees the load is spread among the peers in each level, the forward responsibility of particular node decreases therefore the overall scalability increases (Birrer, 2008). The connectivity degree of each switch at each level is the same in most fat tree variants and since the connectivity degree of the switching elements in Znode and Super node can differ in each level makes it even more scalable than traditional fat trees. In this section, we show through simulations how the Znode can be scaled by increasing the size of the super node, as shown in the recursive construction of chapter 3, and by expanding the internal configuration of the Znode to accommodate larger number of processing power.

5.6.1. Super Node scalability

Initially, we structure an optimal Znode accommodating 512 processors with the parameters shown in table 4.1, and we then increase recursively the size of the super node as 1, 2, 4, and 8. This gives a total number of processors equal to 512, 1024, 2048 and 4096 respectively. The simulation is carried out for the full range of offered load with random traffic patterns.

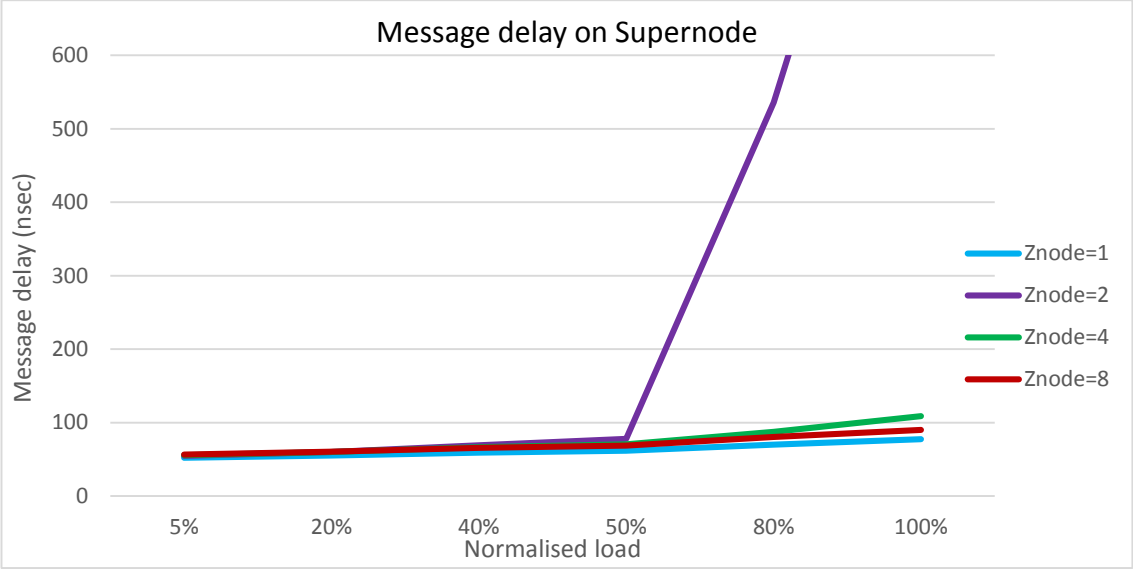


Figure 5.20. Message delay in the scaled super node with 512, 1024, 2048 and 4096 processors

In figure 5.20, we notice that compared to a single Znode, the average delay increases slightly up to a load of 60%. This proves that adding more processors to the network by replicating the number of Znodes into a Snode does not affect the system's overall performance.

Throughput on the other hand, as shown in figure 5.21, steadily increases while the size of the super node expands. This evidently demonstrates that the scalability has little effect on the performance of our system making the super node scalable and the throughput scales as well.

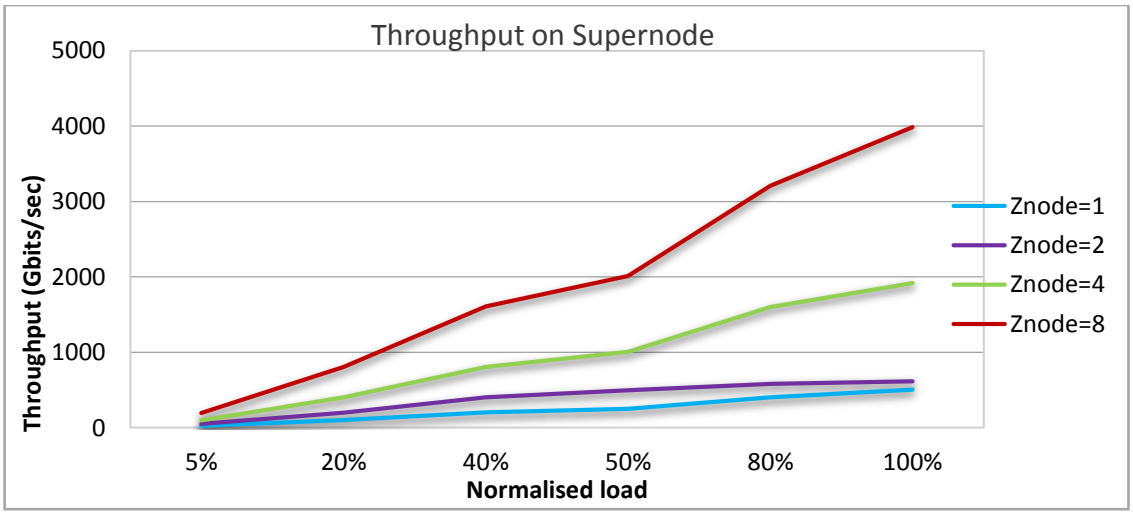


Figure 5.21. Throughput in the scaled super node with 512, 1024, 2048 and 4096 processors

An interesting observation that can be made for random application pattern is that the delay increases when the number of Znodes within the super node is 2 and then starts decreasing and stabilising slowly as the number nodes increases to 4 and then 8. This is due to the fact that when adding more Znodes, the number of side links also increases giving more paths for the side traffic and then reducing the side queuing effect of each routing switch. On the other hand, the relative power consumption decreases as the complexity of a single bar switch to support larger number of processors becomes too unbearable up to a certain number, 4 in this case. It then starts following the extra complexity of the super node, but in all cases it is lower than the threshold of 0 in this specific case.

5.6.2. Znode scalability

The Znode scalability has already been simulated in figures 5.7 and 5.8 of section 5.3.4. It is shown that the throughput scales when more processors are added to the Znode, eventually by re-arranging the zones and the routing elements structure, and finally the delay increases slightly with the increase of the number of processors.

5.6.3. Znode and Snode combined scalability

The scalability can also be achieved by increasing the number of processors in both the Znode and the Snode. Table 5.6 shows the possible combinations between Znode and Snode to accommodate 4096 processors. The following simulations evaluate the message delay and throughput for an offered load of 80% with random traffic. The case of 4096 processors per Znode has already been studied as the Znode scalability. On the other hand, the case of 512 processors per Znode has been addressed in the Snode scalability where we replicate the Znode 8 times to accommodate 4096 processors within the Snode. In this section we simulated the combination of 2048 processors per Znode requiring 2 Znodes within the Snode (2*2048 = 4096). We further investigate the combination of 1024 processors per Znode requiring 4 Znodes per Snode (4*1024 = 4096) and it is apparent that the combination of 2048 uses less complexity than the combination 1024.

Processors per Znode	Znode (m)	Zones (4096 processors):				Routing elements				Links				Relative power
		1	2	3	4	1	2	3	4	1	2	3	4	
4096	1	16	4	4	16	2	24	96	192	18	8	8	16	-11.47
2048	2	8	4	4	16	2	24	96	384	21	9	9	17	-11.56
1024	4	4	4	4	8	6	12	48	192	13	15	15	15	-7.83
512	8	2	4	4	16	1	6	24	96	21	9	9	17	-8.73

Table 5.6. Parameters for Znodes and Snode combinations to support 4096 processors

The tests were performed under Random traffic pattern on normalized offered loads between 5-100%.The configuration's parameters for each set of processors can be illustrated in table 5.6. It must be noted that the higher the number of Zoned nodes are the higher the relative power consumption will be. The difference in relative power is approximately +2 Decibels as Znode increases. As illustrated in figures 5.22 – 5.23, the message delay decreases dramatically as the number of Znodes increases and the throughput increases proportionality making the combined scalability a superlative way of scaling the system.

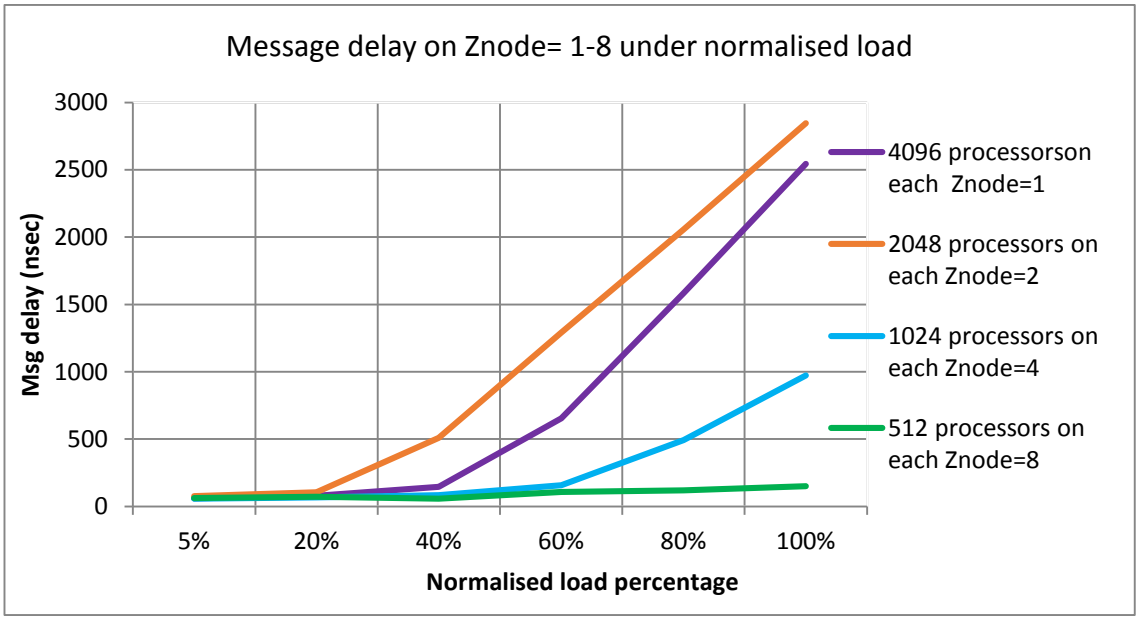


Figure 5.22. Message delay on m=1, m=2, m=4, and m-8

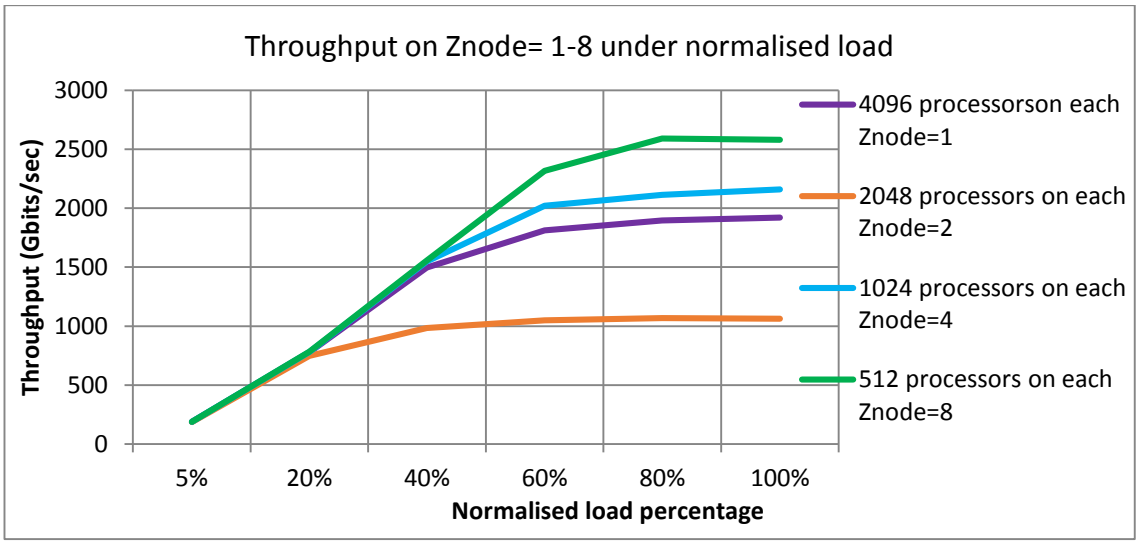


Figure 5.23. Throughput on m=1, m=2, m=4, and m-8

5.7. Capacity

The capacity of the network can be increased by increasing the degree of connectivity or the number of layers. This increase also maintains a fault tolerance system, at the expenses of extra complexity and thus bigger power consumption. In this section we simulate the effect of increasing the number of layers, and hence increasing the capacity of the Znode. We simulate a Znode with 2048 processors of the following parameters; 4, 4, 4, and 32 zones, and 1, 4, 16, 64 routing elements spread over 4 levels. We set a constant inter arrival time of 100 nanoseconds under random application pattern. The results are shown in figure 5.24 and it is noticeable that while the number of layers increases, message delay decreases and throughput increases proportionally. An important outcome is that the layers based on the number of processors can only be increased up to a specific value, where the maximum number of layers is 8. Any higher number of layers than that will begin to have a negative effect on throughput and this can be justified as layers can be "too much of a good thing". The power consumption results are illustrated in figure 5.25, where we notice a trade-off between power and latency. As number of layers increase the latency decreases at the cost of the power consumption which increases proportionally. When we consider up to layer 8 as the best possible layer value for this scenario, we notice that up to that layer the increase in power consumption is not unbearable.

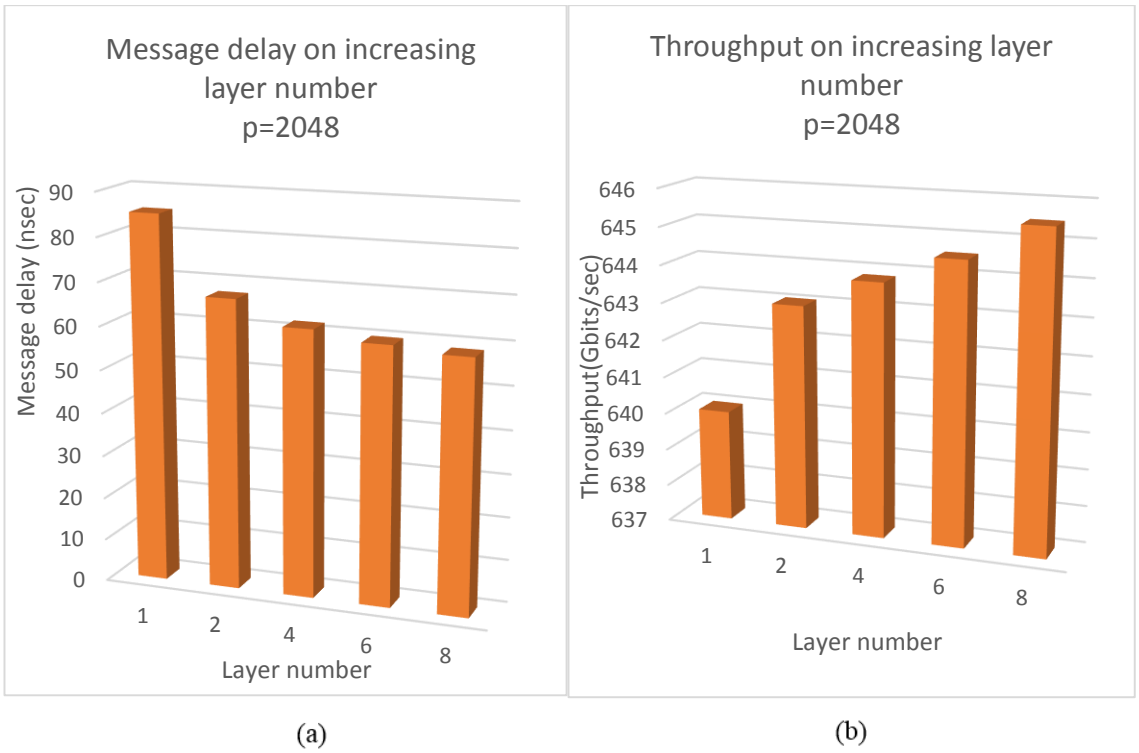


Figure 5.24. a) Latency against increasing layers, b) Throughput against layer number

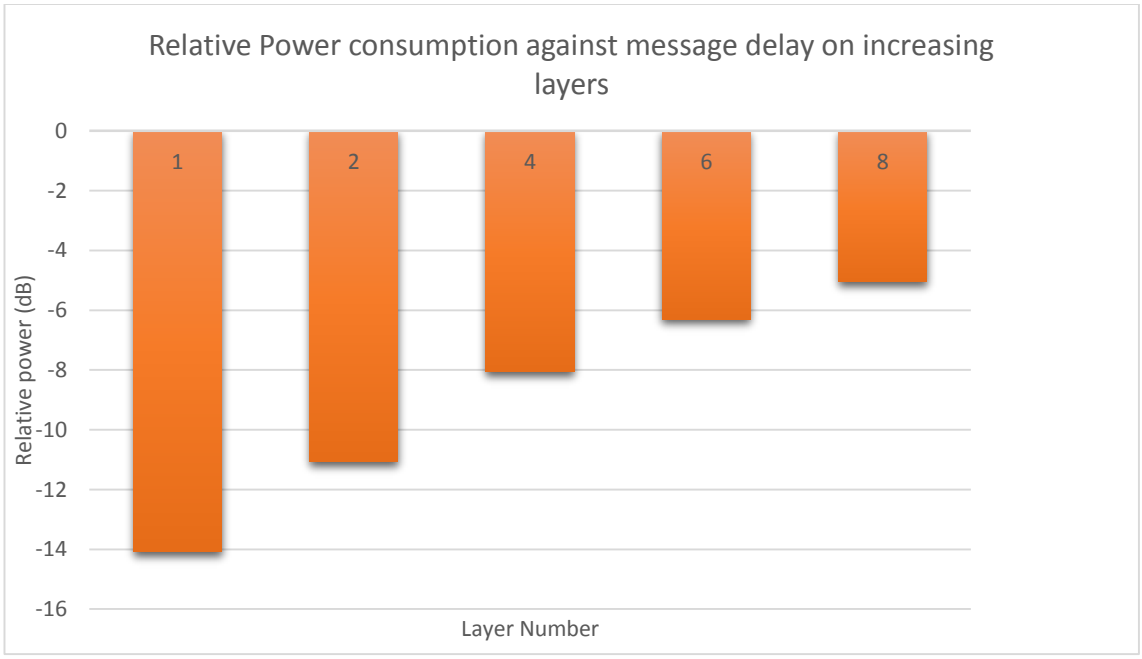


Figure 5.25. Relative Power consumption against Latency on increasing number of layers

5.8. Zoned node architecture against similar fat tree topologies

Znode architecture and XGFT (Source destination addressing) were evaluated in a configuration of 3 levels consisting of 4, 4 and 32 zones for level 1 to level 3 respectively, with a total of 512 processors. Due to the permutation limit in K-ary n-tree architectures, the configuration is slightly different. K-ary n-tree (source addressing) was evaluated in a 3-level configuration, with 8 zones in each level in order to comply with a total number of 512 processors. All tests were performed under a link rate of 1Gbits/sec under Random traffic pattern and virtual cut-through routing was used with a buffer size of 2. Evaluation of Zoned node behavior under various buffer sizes is illustrated in chapter 6. XGFT is simulated under source-destination address routing, k-ary n-tree under destination address routing and Znode under single-bit sliced address routing.

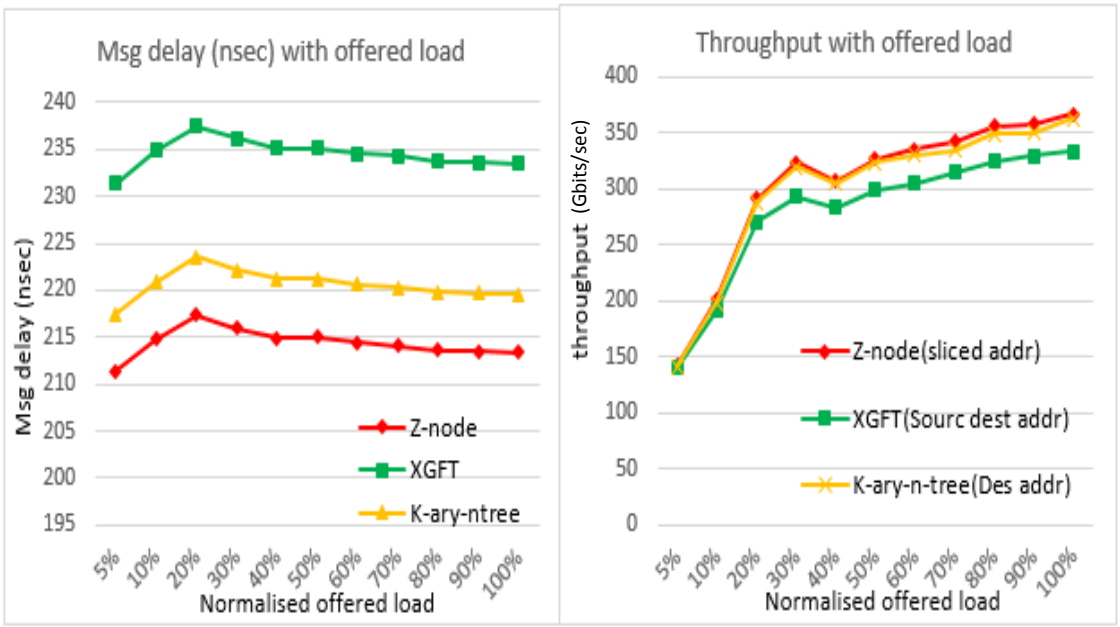


Figure 5.26. a) Message delay against addressing schemes under various applications b) Throughput against addressing schemes

For all the offered loads, XGFT and k-ary n-tree have significantly higher message delay than Z-node (slice addressing). The reason behind it is the fact that K-ary n tree has the same amount of switching elements subsets on each level (fixed arity switches) meaning fewer subsets exist near the root, minimizing the routing choices for incoming packets. While for XGFT the high latency is a result of the source and destination addresses that

are carried along throughout the message's transmission, increasing the overall transmission time. The difference in throughput between the different architectures is less noticeable (Figure 5.26 b), and mainly due to the overall message length where the addresses are counted as part of the throughput.

An overall evaluation of the three architectures is presented in figure 5.27 and 5.29, where the architectures were tested with processors from 128 to 2048 under 50% of load. Zoned node architecture performed significantly better in all processors scenarios (figure 5.27) with lower relative power consumption for processors higher than 1024. In processors lower than 1024 we notice a trade-off between power and performance.

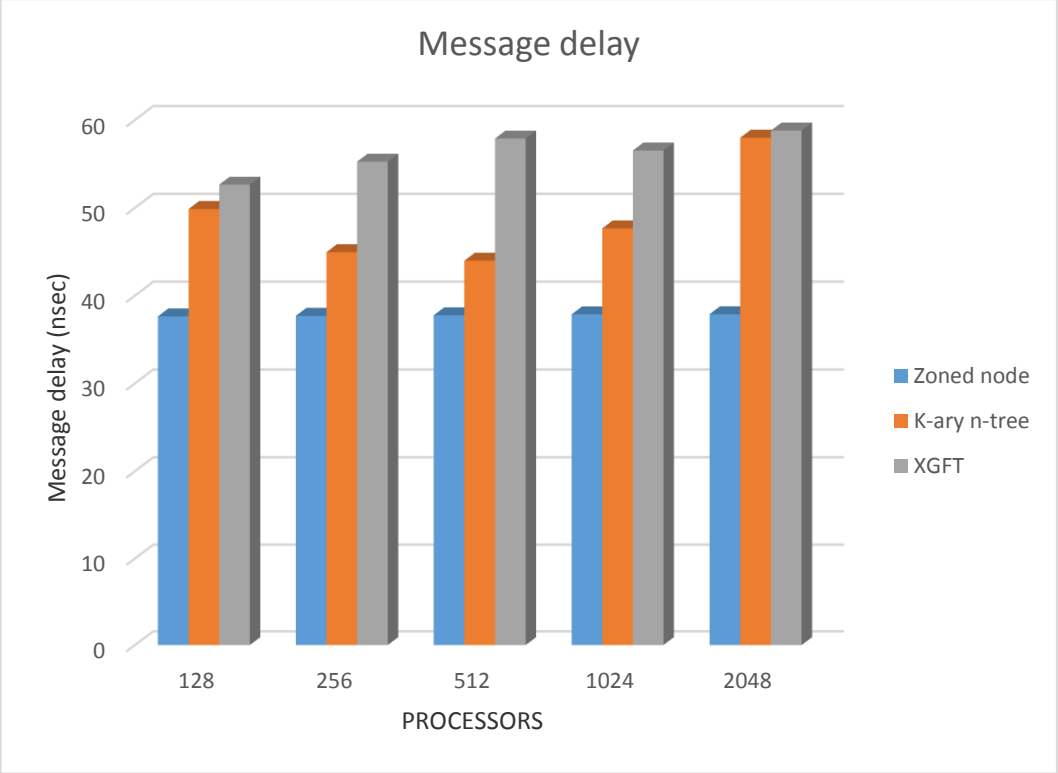


Figure 5.27. Message delay on zoned node, K-ary n-tree and XGFT

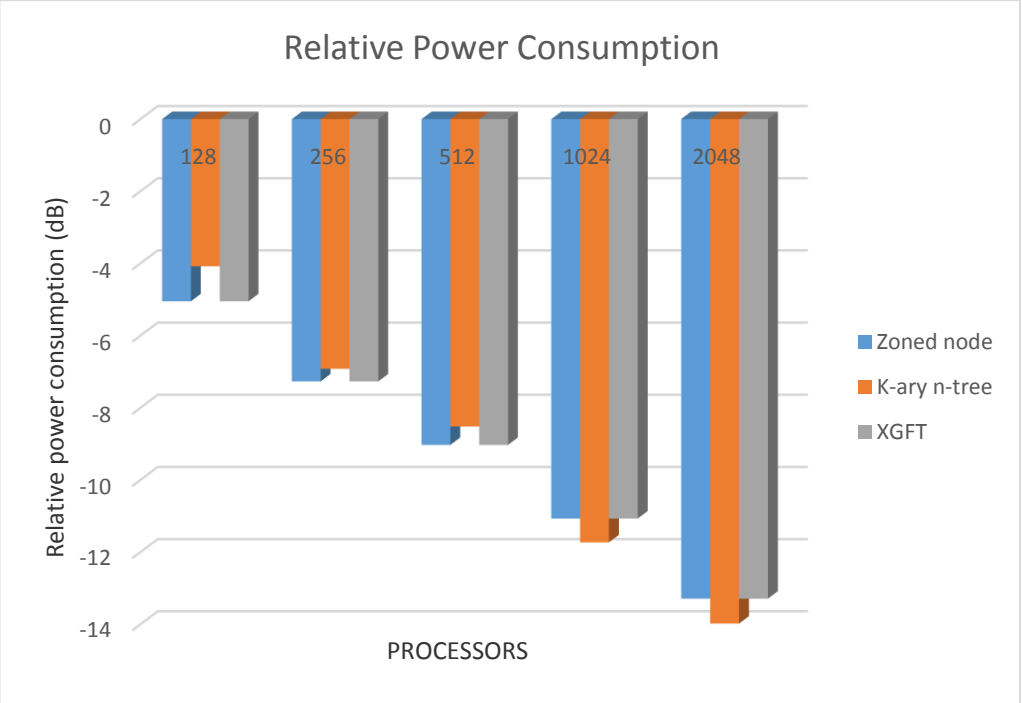


Figure 5.28. Relative power consumption on zoned node, K-ary n-tree and XGFT

Furthermore, the proposed sliced addressing for Z-node architecture provides a faster transmission, especially on high traffic due to the use of the routing bit instead of the destination address. The results shown in figures 5.26 and 5.27verify that Z-node is more stable even on high traffic inputs.

5.9. Summary

In this chapter we have evaluated the proposed Znode and Snode in great depth and detail. Our evaluation encompassed a wide spectrum of parameters, including various processors numbers and application traffic patterns. Unlike other fat tree architectures that have constant level, our proposed Znode configuration shows that the optimal level can be tuned to accommodate larger number processors without losing performance. We have shown, unlike other topologies such as XGFT, it is possible to find an optimal configuration among the fat tree classes that performance better than others in term of message delay and throughput.

Zoned node can provide efficient fat-tree class topologies, while resolving their weak scalability, along with an addressing scheme that can minimise the network latency. The

results have confirmed the performance enhancements that can be obtained under Zoned node architecture, as significantly lower latency was achieved compared to other similar architectures. Routing addressing schemes including source-destination, destination, flat addressing and sliced addressing were evaluated based on their latency and throughput. A comparison of those addressing schemes clearly shows that the sliced addressing exhibits the lowest latency and the highest throughput. This chapter provides convincing enough results that the zoned node is a flexible and promising architecture that can offer various advantages over conventional fat tree topologies.

Unlike other, Znode provides a much flexible scalability without major upheaval on the performance. We have shown that slight message delay is encountered for when the number of processors in the system increases, while the throughput alongside scales linearly. Furthermore, unlike other topologies in the same class, Znode increases it capacity by adding more layers, which can also provide a mechanism for fault tolerance.

In the next chapter we have an overview on possible implementations of the Znode and Snode, but looking at the ways to implement routings and address translations in hardware

CHAPTER 6. IMPLEMENTATION

6.1. Introduction

This chapter focuses on suggesting and identifying the key hardware elements that can be used to implement the various algorithms and protocols of the Znodes and Snode. It presents a general implementation overview of the architecture, highlighting the features that enable the physical implementation of Znode. As explained earlier for the purpose of this research a parallel processing simulation was implemented in C++ called Galanet that can simulate multiple Zoned nodes as a cost effective solution to avoid physically implementing the system. This chapter analyses both the software and hardware implementation of the proposed architecture and discusses any issues that might be faced during a physical implementation.

6.2. Network configuration, initialisation phase

The Znode network needs to be initialised during the configuration phase before it can be used. The configuration phase involves the automatic configuration of the number of the processors, the routing elements per zone, the interconnection links per routing element, the number of zones, the number of layers and finally the number of znodes. The most important parameters during the configuration phase are the mapping of the physical address of the treads and hence processor into switches port labels. This is achieved in general by equation 5.1. The size of the port labels plays an important role on the performance and latencies of the whole network $\log_2 p_i$. There are two ways in which this equation can be implemented. In software, a configuration file will contains the address mapping between the thread physical address and the port labels, and on request an address is converted into set of port labels. This is predefined initially as the configurer is aware of the network structure. In hardware, this will be explained in the following sections.

$$p_i = \left(\frac{X}{z_1 \times z_2 \times \dots z_{i-1}}\right) \% z_i \qquad (6\text{-}1)$$

Where p_i the port label at is level i that leads to the destination processor, X is the physical address of a processor and z_i is the number of zone in level i. It needs to be mentioned that z_0 will always be equal to 1 for convenience. For further explanation refer to chapter 3, section 3.6 and definition 1. The Equation 6.1 can be extended to evaluate the port labels for the Snode, by including the parameter m (number of Znode) in the place of z_i, which theoretically defines another zone extension as shown in equation 6.2, where m is the number of Znodes in a super node.

$$p_m = \left(\frac{X}{\prod_{i=1}^{n} z_i}\right) \%m \qquad (6\text{-}2)$$

6.3. Switching elements' internal architecture

The Switching elements in out tree configuration can be implemented in various ways, but the preferred one is crossbar design due to its simplistic features along with its non-blocking nature. Crossbar switches can be implemented depending on the preferred buffering stage whether is internal or output-buffered. The routing switches are scalable. The parameters that justify the switch architecture is the number of input and output ports, the buffer size, along with the width of the channels. The switching nodes have three primary functions which are routing, switching and arbitrating.

The routing switch as illustrated in figure 6.1 has three different routing directions the upwards routing the downwards and the side routing. The multiplicity in the upwards output ports enforces the adaptive routing.

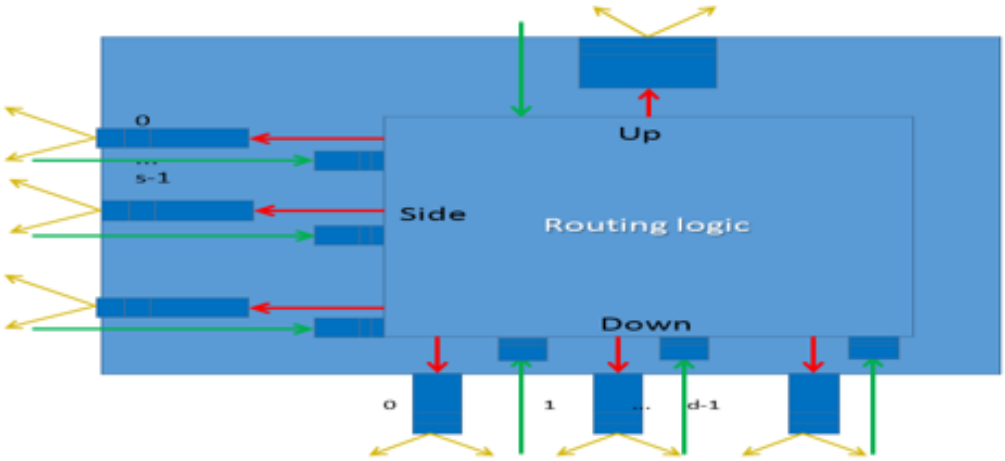


Figure 6.1. Routing switch port logic

In flat addressing a message that comes in from the lower input port goes straight to upper output port, by selecting any available port, without any decision making. Once the message reaches the root switching element it will start moving downwards or sideway, where a decision will be made to find out the lower output port to forward the message to.

In sliced addressing a message that comes from a down port, depending on its routing fields, might move either to a down output port following its port label after reaching its common ancestor level, towards upper output ports adaptively without any decision routing or side output port according to the address label, as shown in figure 6.1. The detailed logic of the routing element for sliced addressing on the other hand can be illustrated in figure 6.2.

Once a message arrives from the upper input port, the destination label/address is extracted and recorded in the label register; with the use of additional hardware it will be decoded and checked to identify the proper lower multiplexer to get forwarded to and from there to the required down output port.

When a message arrives from the side input port its destination address is extracted and then stored in the side label register; with some additional hardware it will be decoded and since the only option is to move downwards it will then be forwarded to the appropriate lower multiplexer and to the proper output port.

A slightly more complicated process exists once the message arrives from the down input port. In that case, the routing bit (ρ) is stored in the flip flop, and then a decision is made based on the routing bit value. If the routing bit is equal to 1, the message gets forwarded to the upper multiplexer from where is transmitted out of the proper upper port adaptively selected. On the other hand, if the routing bit is equal to 0 then the consecutive message destination label is extracted and stored in a label register and then checked against the SID (Side Identity Label of the switch) that specifies the identity of the Znode which is an element of the set {0,1, …, m-1}. This check is carried out in order to find out whether the message has to be forwarded to the next Znode in the super node environment or stays in the same Znode. If the SID's is the same as the message port label then it gets transmitted out of the lower multiplexer, using the next port label; if the labels are different, the message gets transmitted out of the side multiplexer to reach the identified

Znode, specified in the port label of the message. In case of a single Znode the SID is not checked and the message gets forwarded to the lower multiplexer, in all cases.

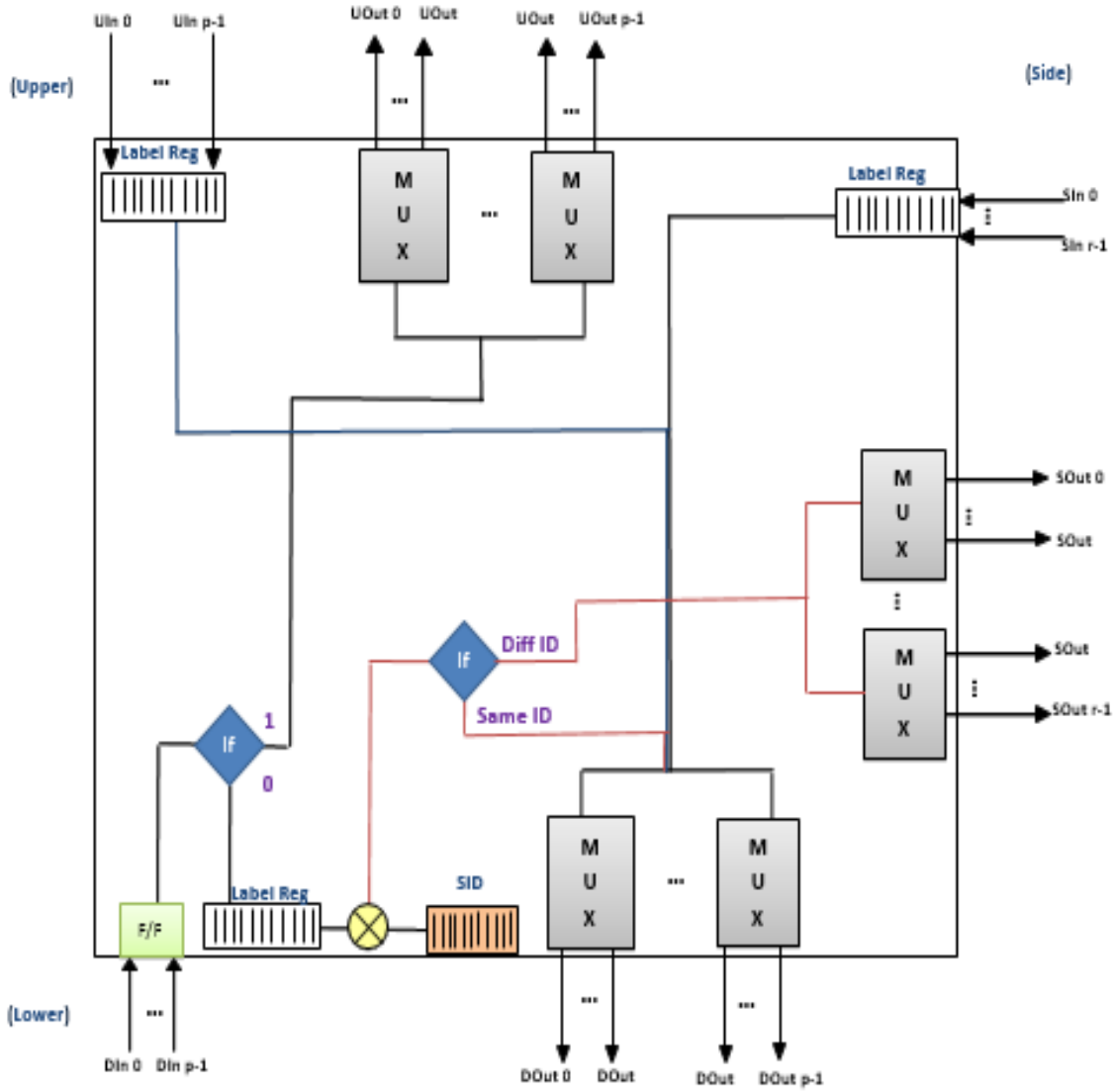


Figure 6.2. Detailed switch port logic

A conflict or collision occurs, when messages coming from upper levels, side nodes or common ancestor level aiming for the same destination multiplexer. The switch select arbitrary a message and allocate resources to it while messages involved in the conflict are blocked momentarily. In case of wormhole switching implementation, blocked messages are buffered on the links, until if necessarily the memory of the processors is available. When the output multiplexer is freed, they will progress consecutively. In case of Virtual Cut through (VCT) switching implementation, messages are stored in the buffer of the multiplexer that is busy. In the case of buffer overflow, messages competing for

the same resource are automatically buffered in the links. This is a unique switching strategy adopted by Znode architecture where both hybrid switching operations between wormhole and VCT are complementary deployed. This scheme based on request to send concept prevents messages from being dropped when buffered are full, and hence eliminate the need for a retransmission which is not always a desirable option for high speed parallel systems. Performances of this scheme are shown in figure 6.8 – 6.14.

6.4. Sliced algorithm

The usage of sliced algorithm minimises the buffer space requirements and the communication latencies compared to other well-known standard approaches. In sliced addressing, the address is cut into multiple slices that the interface can translate as illustrated in figure 6.3. In the hardware implementation, two operational phases exist in the interface layer. The interface has to be aware of the network configuration such as the number of zones, the number of processors and the number of switches and ports per level. The important question here is how do we map the network configuration to the interface layer, how would the interface learn the configuration.

The first operational phase of the interface layer is to manually set up the interface by changing the jumper pins. As an example let's take zone1=8, zone2=4 and zone3=8, which would translate to latencies 3, 2 and 3 respectively. For each zone one would respectively set up the jumper pins to reflect that in the interface. An automatic option would be to have a software configuration file, which can be uploaded into the controller of the interface during the installation phase, so that the interface can read it and reconfigure itself appropriately. A reconfigurable hardware can also be used as a solution. The hardware will automatically change the connections as required. The latency configuration is therefore used to extract the port label in hardware from the physical addresses.

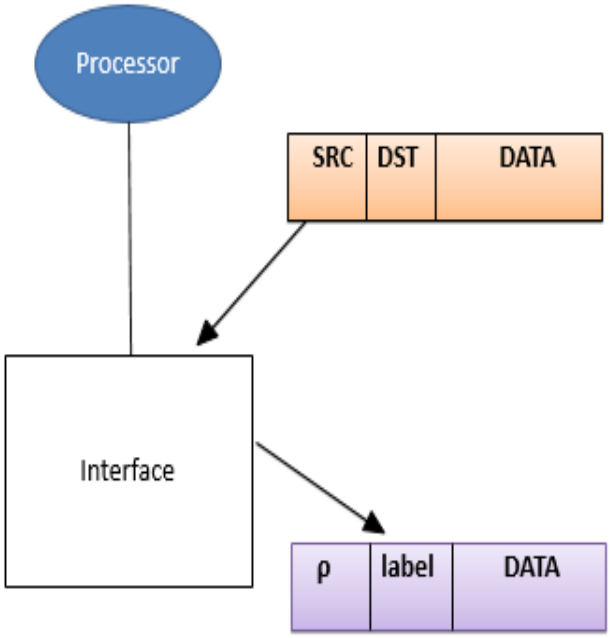


Figure 6.3. Network interface address translation

6.4.1. Address translation:

The addresses need to be translated from logical addresses that are generated by applications and then translated into physical addresses that map directly on the boundaries of the label fields of the zones into port labels. Firstly a logical address is converted into a physical address using equation 6.1 or 6.2, and equation 6.3 below; this may be carried out during address translation at the application or configuration phase, in the processor. The physical address is then converted automatically through the address register in the interface, as shown in figure 6.4. The phase of address translation is not required when the physical address is power of 2 (2^n).

$$\mathrm{P} = p_1 + \sum_{i=1}^{n} p_{i+1} 2^{\Sigma_{j=1}^{i} \lceil log_2 z_i \rceil} \tag{6-3}$$

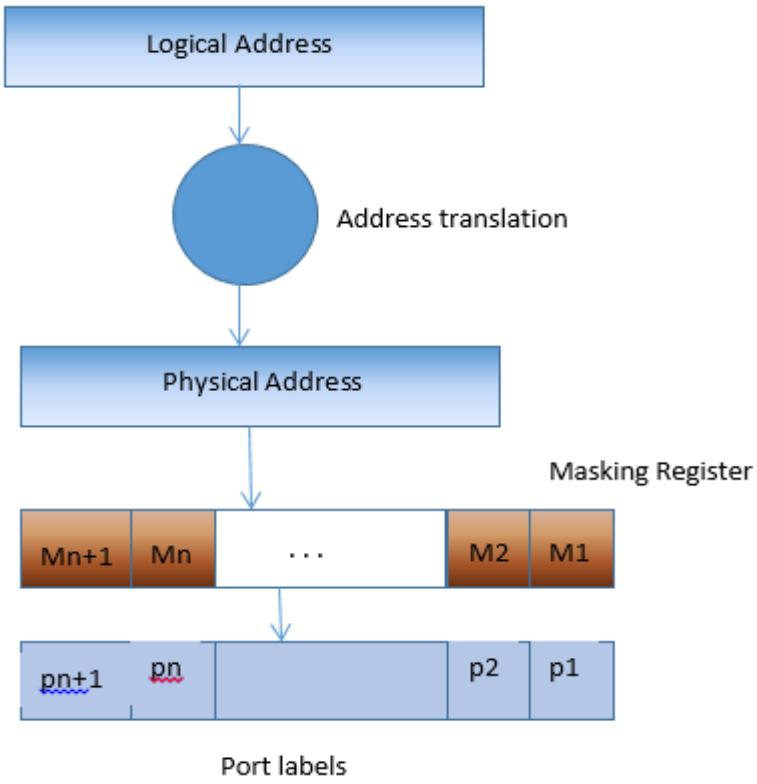


Figure 6.4. Logical address translation into port labels

Two examples in table 6.1, for a super node containing 2 Znodes (m=2), and a Znode has 3 levels (n=3) and zones z1=3, z2=2, and z3= 3 in the first example and zones z1=4, z2=4, and z3=2 respectively, show the translation of a logical address 26 into a physical address 42, and then into the same address 26. This indicates that when the zones are power of 2, the logical address is the same as the physical address, and no translation is required. The reconstruction of the logical address simply proves using the weights of the zones that it is equal to the original logical address. In fact the port labels (1;1,0,2) can be extracted from the logical address, but then arranged as physical address to be extracted by the hardware of the interface using the mask register as shown in figure 6.4.

Physical Address	Port labels (eq. 5.1, 5.2)	Reconstructed logical address	Physical address (eq. 5.3)
26	26/3 = 8, 26%3=2 8/2=4, 8%2 = 0 4/3=1, 4%3=1 1/2=0, 1%2=1 (1;1,0,2)	2 + 0*3 + 1*(3*2) + 1*(3*2*3) = **26**	2 + 0*(2^(2)) = 0 + 1*(2^(2+1)) = 8 + 1*(2^(2+1+2)) = 32 = **42**
26	26/4 = 6, 26%4=2 6/4 = 1, 6%4=2 1/2 = 0, 1%2=1 0/2 = 0, 0%2=0 (0; 1,2,2)	2+ 2*4 + 1*4*4 + 0*4*4*2 = **26**	2+ 2*(2^(2)) + 1*2^(2+2)) + 0*2^(2+2+1) = **26**

Table 6.1. Example of address translation from 26 to 42

6.4.2. Routing bits extraction:

The physical addresses of the destination and sources are further compared in hardware at the interface to extract the routing bits of the sliced bits algorithm described in chapter 4, in the addressing section. Recall that if the routing bit is 1, the message keep going up the tree, it will reach it common level, when the routing bits indicates a 0. Therefore, this source routing algorithm extract the information of the common level before the message leaves the interface. Figures 6.5 shows how the routing bits are extracted with a set of comparators, which simply implement the definition 4 in section 4.3.

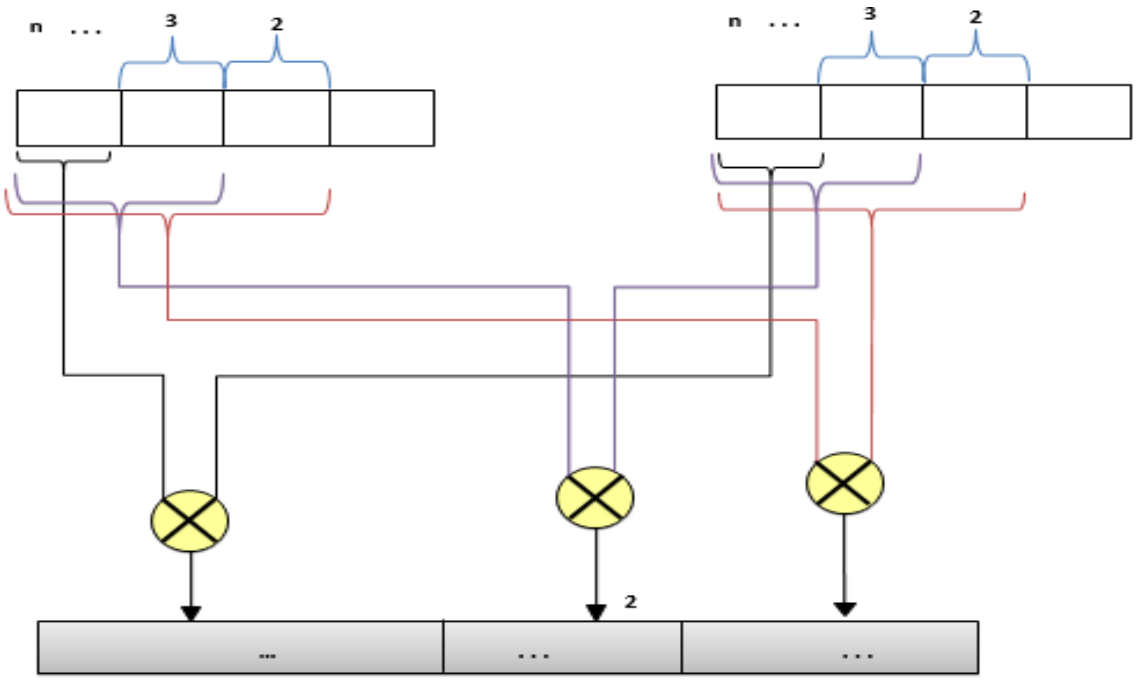


Figure 6.5. Detailed address shifting

6.4.3. Message structure

The routing and addressing information extracted by the interface are carried along with the message, as shown in figure 6.6. The message format contains the routing bits until the common level l, (bit=0), and then the side direction (pn+1), and then the down direction with from the common level to the leaf processor (pl to p1), followed by the payload (data).

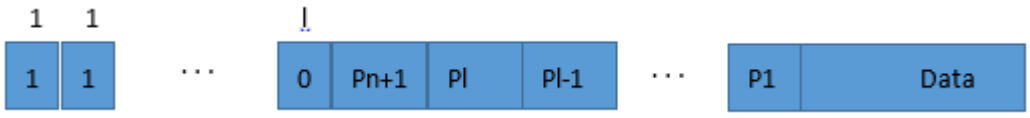


Figure 6.6. Flit/label representation

6.5. Buffering issues in the Z-Node configuration

In this section we explain the switching operation of switching elements. Simulation compares the performance for several application patterns under the switching operations of Store-and-Forward (SF), Virtual cut-through (VCT) and wormhole (WH) schemes. As shown in figure 6.7, each multiplexer has a queue associated with it, depending of VCT or SF. If no buffer is available the WH is implemented automatically. Finally with limited

buffer in VCT, a hybrid implementation is realised, where some messages are buffered in the queue and others are kept in the links. The flag in the figure 5 .7 indicates if there is a room to accommodate the requested message or not.

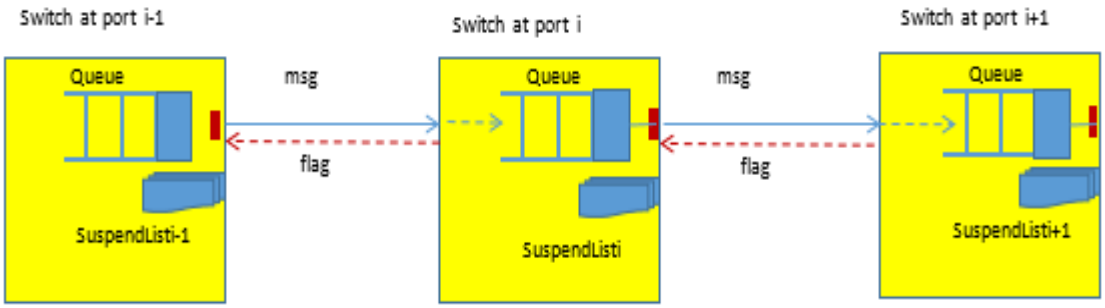


Figure 6.7. Buffering in the routing switches of the Znode

We will explain how the switch operations store-and-forward (SF), virtual cut-through (VT) and wormhole (WH) routings are affected by the size of the buffer and the application patterns. These routing algorithms are well known in the literatures and have been documented intensively for various networks (Dally 2004), (Hennessy, 2012). Although the trade-off of the Z-node is to adopt the sliced-flit routing, suggested in this thesis, on the top of wormhole scheme, it is important to synopsis the impact of the buffering size of each switch across the spectrum of the most popular routing algorithms as it influences the overall cost of the system. In all the following diagrams the network request time is considered as normalized by the ratio of actual response time against transmission time. While the message delay is the normalization of message delay against transmission, so for example if the normalized message delay is 1 then the actual message delay would have been 32 nanoseconds. The buffer sizes illustrated in figures 6.8- 6.14 stand for the number of packets the buffers can store, for instance if the buffer size is equal to 1 it would mean than 1 message can be stored. The simulation in this section uses a super node of 4 Z-nodes of three levels each with the following parameters R1=2, R2=4, R3=8, z1=4, z2=2, z3=4, ψ_1=1, ψ_2=2 and ψ_3=2.

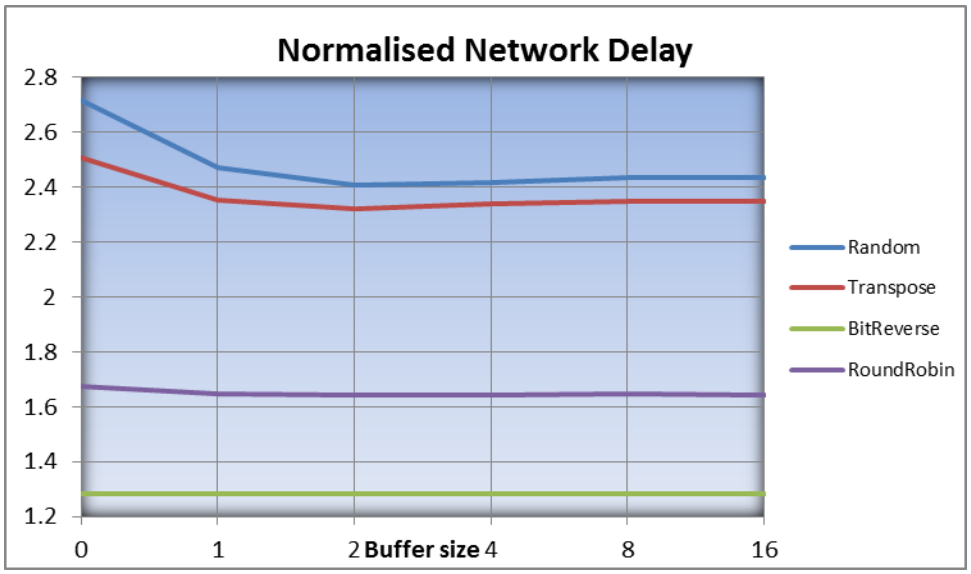


Figure 6.8 Normalised Network Delay by transmission time under VT routing for application patterns in respect to switch buffer sizes – message number.

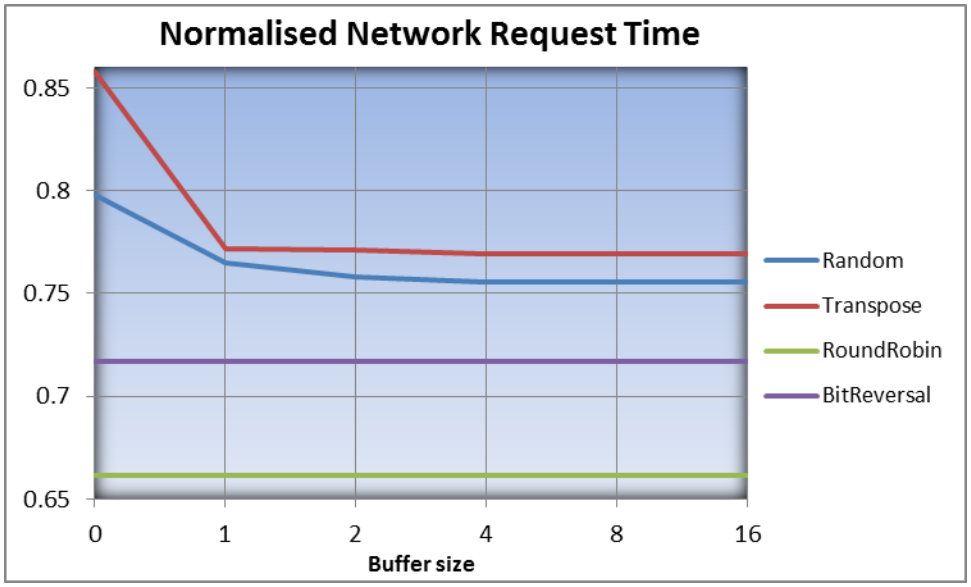


Figure 6.9. Normalised Network Request time by transmission time under VT routing for application patterns in respect to switch buffer sizes – message number.

Figure 6.8 and 6.9 shows the normalised network delay and network request time for various traffic patterns, under different buffer size. The network request time includes the time from when a thread has issued a message ready for a transmission until the message is granted access to the network through the interface of the processor.

Notice that wormhole operation is obtained when the size of the buffer is zero under the virtual-cut through mechanism. In this case, messages that cannot progress due to

unavailability of resources (free ports) are suspended in the physical links. On the other, hand for virtual cut-through uses physical links and queues within the switches to buffer messages that cannot progress. As shown in figures 6.8 and 6.9 with an exponential inter-arrival time giving an offered load of 80%, with a message length of 32 bits, round robin and bit reversal traffic seem to be inert to buffer size whereas random and transpose traffics exhibit slightly higher delay for zero buffer size. The network request time follows, as there are less buffering in the overall network. Therefore, messages produced by threads are kept in the memory, until the interface ports become available.

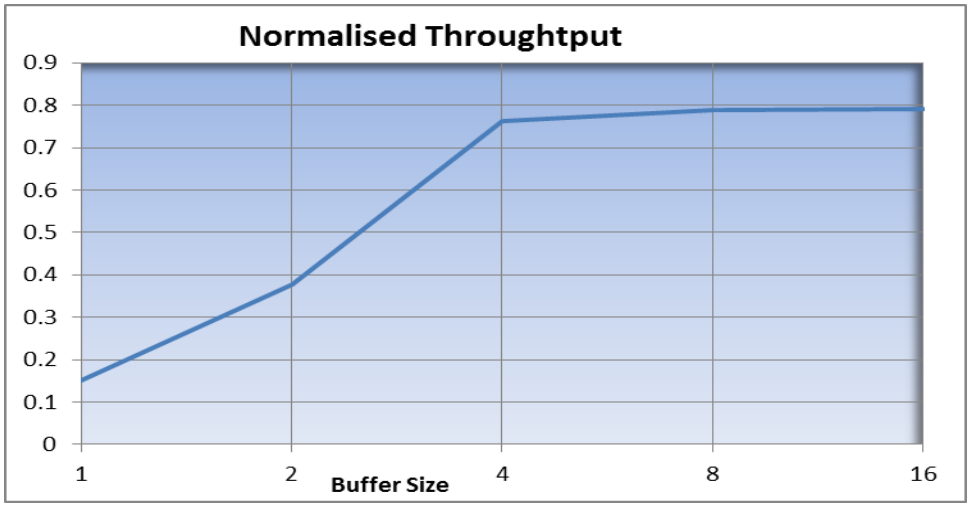


Figure 6.10. Normalised throughput by the network capacity with SF operation, under various buffer sizes – by message number.

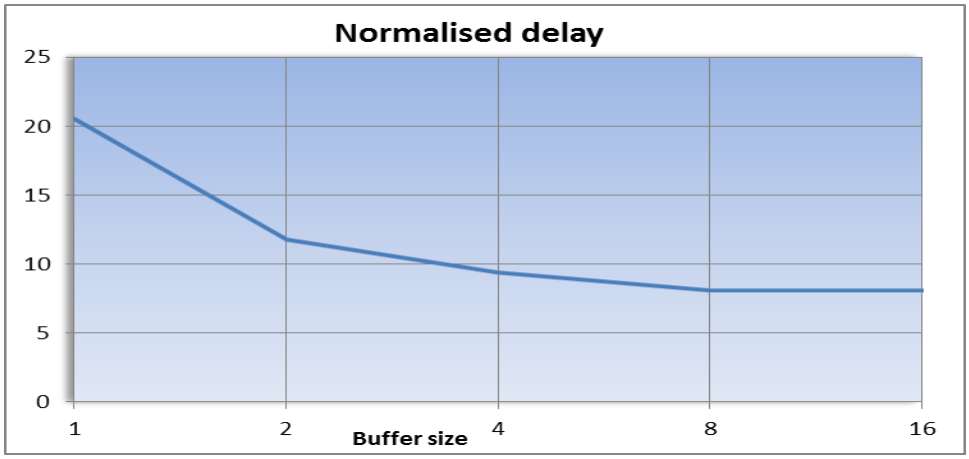


Figure 6.11. Normalised delay by transmission time with SF operation, under various buffer sizes – by message number.

For the store-and-forward routing, as shown in figures 6.10 and 6.11 with random traffic, the delay although is worse than VT for all offered loads, it improves with the buffer size

and so does the throughput. SF relies on more storage to maintain messages that cannot progress to their destinations, and by default the routing also imposes such a strategy to store messages before forwarding them to the next available port. This is quite an expensive solution to implement while the delay remains high compare to VT. Store-and-forward routing is not used in modern and fast network for supercomputers. However it is still adopted in some switches for Internet connections, although faster switches implement versions of VT.

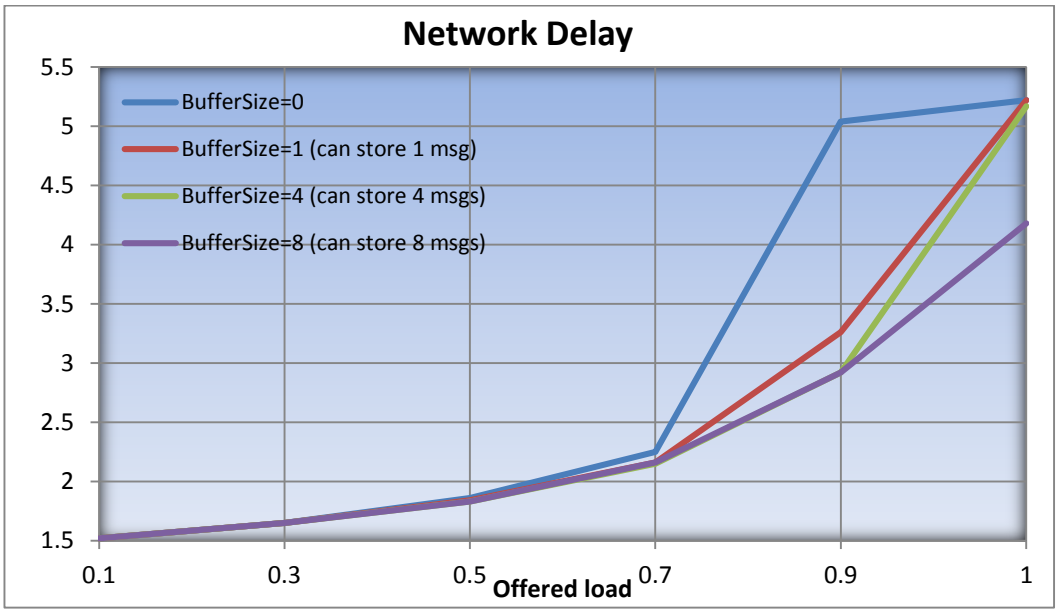


Figure 6.12. Normalised delay by transmission time for random traffic, under various offered loads with different buffer sizes – by message number.

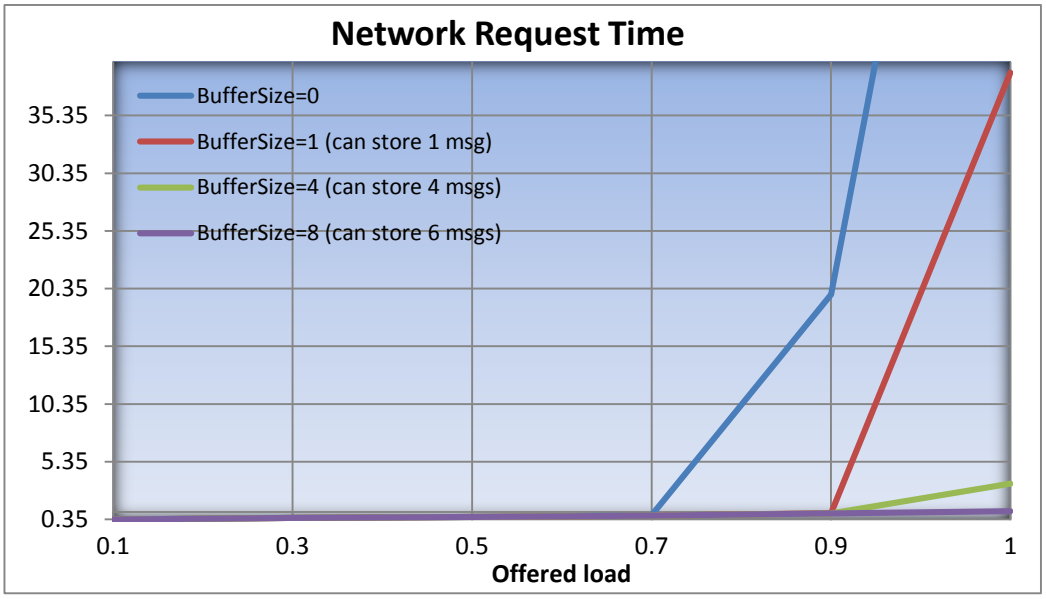


Figure 6.13. Normalised delay by transmission time for random traffic, under various offered loads with different buffer sizes – by message number.

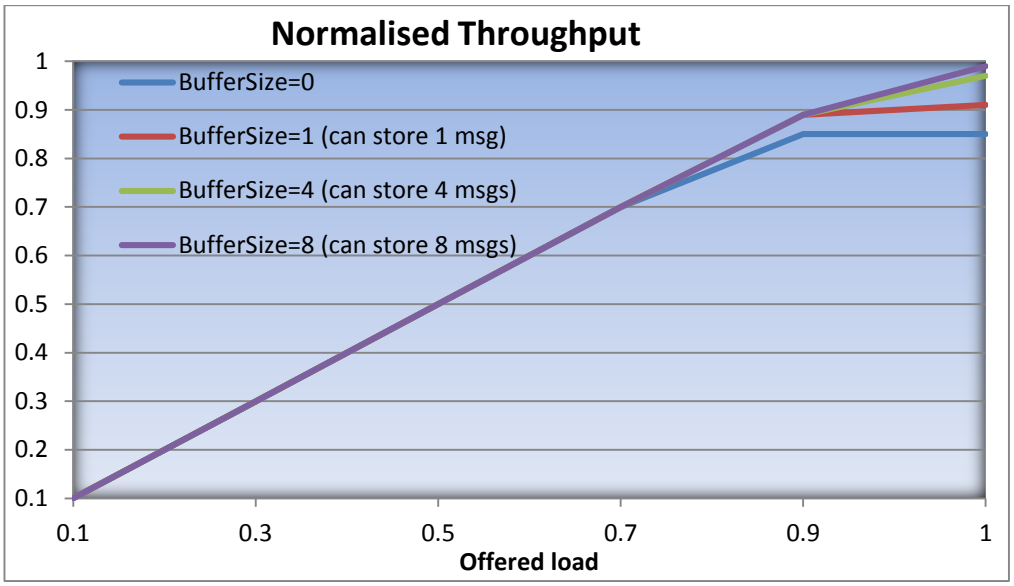


Figure 6.14. Normalised throughout by network capacity for random traffic, under various offered loads with different buffer sizes – by message number.

Figures 6.12, 6.13 and 6.14 show the normalised network delay, network request time and the normalised throughput for random traffic under various offered loads with different buffer sizes for VT routing. For up to 70% of offered load the buffer size does not impact much on the network delay and network request time. At higher offered load, eventually more buffering improves the throughout and the network delay. For wormhole, messages are kept in the memory for long, before they are granted access to the network. This is due to the lack of buffer in the overall network. Despite this limitation, wormhole routing is the most popular routing for high performance network. In this research both wormhole and VT switch operations are supported with our proposed sliced-flit routing algorithm. VT however will required extra complexity per switch to store and manage messages.

6.6. Summary

In this chapter we described the implementation details of Znode architecture along with the hardware requirements to efficiently implement Znode. Particularly the switching elements initialisation and operations where defined in detail. Additional hardware that might be required that is not covered in this thesis, is due to the widely emerging technology of electronics, where more than one solutions can be applied. We provided diagrams to illustrate the feasibility of the design, a simple logical approach with

comparators, registers multiplexers and flip flops, but the exact details of the design will be determined by the current technology.

CHAPTER 7. CONCLUSION AND FUTURE WORK

7.1. Conclusion

This thesis has given a detailed analysis of our proposed Znode and Snode architecture, and emphasised on the topological advantages and features along with the connectivity utilised. The impact this work has is the generalisation of the fat tree classes in term of topological design and address routing adaptation. This provides a new ground on how to perceive the structure of fat tree classes and consider them as building blocks for more complex networks that will accommodate larger number of processors. The general semantic we defined in this work for the Znode can be expanded to identify fat tree classes, and hence opt for the best configurations that optimise the relative power consumption and maximum performance gain. Our proposed Znode structure introduces an architecture that can identify efficient fat-tree class topologies, by resolving their weak scalability, along with an addressing scheme that can minimise the network latency.

7.2. Summary

The topological advantages of Znode architecture have been fully defined throughout this thesis. We have proposed several new aspects that do not exist in any other fat class topologies such as the optimum configuration. The optimum level for each set of processors has clearly better performance even on high offered load. This proves that higher number of processors accommodated by a single Znode require higher levels to arrive at a better performance. A comparison of the optimum configuration against a non-optimum k-ary n-tree and XGFT through simulation indicated that our proposed Znode has better performance in terms of message delay and throughput.

Backward and forward connectivity along with connectivity degree can increase the routing paths without increasing the routing elements, to achieve the requirement of a fat tree. Which according to our results better performance is obtained under lower number

of routing elements and higher number of connectivity degree. In both positive and negative connectivity, when the connectivity degree is higher, the latency is reduced, at the price of slightly higher relative power consumption. Negative connectivity performs better than positive connectivity for some cases as the routing elements increase toward the lower levels and hence deterministic traffic experience less queuing and waiting times. But this is achieved at the expenses of higher complexity.

Our proposed address routing algorithm, single-bit sliced address, that uses bit stream to find the direction for the messages has the lowest latency compared to flat, source-destination and destination routing algorithms. Furthermore, it was identified that in all application traffic patterns, the single-bit sliced addressing performs is better than other conventional addressing schemes. The latency decreases noticeably under sliced addressing due to the single bit per hop routing as opposed to carrying along the entire source and the destination addresses.

Compared to other fat tree class architectures k-ary n-tree, and extended graph (XGFT) it was identified that Znode provides significantly increased performance especially on higher offered load where other topologies suffer. The reason behind that is in Znode the number of routing elements is balanced against the degree of connectivity to create more paths for messages to flow towards their destinations.

This thesis also presented various ways of scaling both Znode and Snode, by increasing the number of connectivity degree offering a stable increase in the supported processors, by increasing the size of the super node recursively or by increasing both the size of the supernode and the size of Znode. It was proven that increasing the number of processors on each Znode does not affect the overall performance to the system, offering a balanced and efficient scalability means. When replicating the number of Znodes into a Snode, the higher the number of Znodes the lower the latency will be; therefore better performance can be achieved under Snode and Znode combined scalability.

The stack layered approach, proposed in this thesis, which refers to layers forming the Znodes structure connected in parallel, increases even further the connectivity and the capacity of the system. As shown in the results of the simulation, 3 layers seem to reduce

the message delay to a point where an increase in the number of layers does not show any noticeable decrease in message delay.

The results have confirmed that the performance enhancements can be obtained with Znode architecture. But this high performance is guaranteed sometimes at the expense of slightly higher complexity and, hence, relative power. *"There is no free lunch!"*

7.3. Future work

Through this work, several aspects and new directions have emerged around the design and development of the Znode. We can identify the following non-exhaustive list.

- Hybrid topologies and multiple super nodes,
- Cross layers connections,
- Fault tolerance, and data centres,
- Exascale computing,
- Multi-core interconnections.

The work has already started, by others, with the development of hybrid topologies that are based on the concept of multiple super nodes with the fundamental base of Znodes. This approach that would include features of various topologies such as a hybrid tree can minimise the long links of fat tree topologies. A good suggestion would be to utilise a different topology type for each super node. This brings new ways of generalising various modern interconnections such as Milkyway, BlueGene, and DragonFly. An example would be for instance 6 super-nodes having full connectivity each, internally with Znodes and then externally adopting any popular interconnections such as torus, hypercube, etc. To extend even further the topology, the super-nodes can be divided into groups with each group connected to other groups of super-nodes, producing hence a complex infrastructure that can accommodate far larger numbers of processors.

In this thesis we have exploited the use of stacked layers as explained in chapter 3. Each Znode can be replicated into many layers, hence increasing the capacity of the system while reducing the latency controlled, and balancing the traffic at the interface point – or injection points of the network. Obviously, this does not consider load balancing further

up the tree. So far only parallel connected layers were utilised, but cross connected layers can be investigated as a further solution to enhance the connectivity degree between adjacent layers and increase the overall performance of the system by balancing the traffic up the tree. Initial work by others has already started in this direction, and the preliminary results show that better performance are obtained at the expense of more power consumption.

Fault tolerance is an important factor that needs to be taken into consideration when utilising large scale systems. A popular but expensive fault tolerance technique used in BlueGene is the implementation of circuits to switch off the malfunctioning element (Jacobsen, 2006). Another approach used in most fat tree classes is the interval routing. However, this approach requires the use of deterministic routing, which seems to be inferior in performance to the adaptive routing that we embraced in this work. A fault tolerance approach can be investigated under the concept of Znode with the development of multi-layers cross interconnections, the use of redundant links – already implemented in the current simulator but not used in this thesis as fault tolerance is not the objective of this work, and degree of connectivity management. Obviously, these come at the price of more power consumption and complexity. For data centre design, these techniques can be adequate as complexity can be weighed against the value of keeping the centre and servers operational for many days, without a major disruption to the whole network.

The perception of the way interconnection networks will be designed is changing vividly over the past few years due to the increase in speed of the processing units (Peta and Exa flops). A recent research trend is exascale computing, which would increase the computational speed of processors but can lead to extreme power constrains (Bekas, 2013). Exascale is expected to minimise the memory per core and per flop which will increase the need of latency control algorithms, as processors will be thousands of cycles away which will increase the waiting time to complete operations thus the overall performance will suffer (Cropp, 2013). The Znode and super node concept can be taken further to develop topologies that can accommodate more processing power with acceptable performance.

Finally, the Znode with possibly hyper-bus or hyper-ring configuration can be considered as another alternative to connecting many core machines together within a single chip. This will give a three phase network design, which is the on-chip, off-chip and on board/rack interconnections.

BIBLIOGRAPHY

Ajima, Y., Inoue, T., Hiramoto, S. & Shimizu, T. (2012). Tofu: Interconnect for the K computer. *Fujitsu Science And Technical Journal*, 48 (3), pp. 280--285.

Ajima, Y., Takagi, Y., Inoue, T., Hiramoto, S. & Shimizu, T. (2011). The tofu interconnect. *19Th Annual IEEE Symposium On High Performance Interconnects*, pp. 87--94.

Al-Faisal, F. & Rahman, M. H. (2009). Symmetric tori connected torus network. *12Th International Conference On Computer And Information Technology (ICCIT)*, 21-23 Dec 2009 Dhaka: pp. 174--179.

Alam, S. R., Athanassiadou, T., Robinson, T. W., Fourestey, G., Jocksch, A., Marsella, L., Piccinali, J., Poznanovic, J., Cumming, B. & Ulmer, D. (2013). First 12-cabinets Cray XC30 System at CSCS: Scaling and Performance Efficiencies of Applications. *A New Vintage For Computing Conference (CUG)* May 2013

Alverson, R., Roweth, D. & Kaplan, L. (2010). The gemini system interconnect. *18Th IEEE Symposium On High Performance Interconnects*, pp. 83--87.

Antelo, E. (2009). A Comment on" Beyond Fat-tree: Unidirectional Load-Balanced Multistage Interconnection Network". *Computer Architecture Letters*, 8 (1), pp. 33--34.

Arimilli, B., Arimilli, R., Chung, V., Clark, S., Denzel, W., Drerup, B., Hoefler, T., Joyner, J., Lewis, J., Li, J. & Others (2010). The PERCS high-performance interconnect. *18Th IEEE Symposium On High Performance Interconnects.*, pp. 75--82.

Bakhoda, A., Yuan, G. L., Fung, W. W., Wong, H. & Aamodt, T. M. (2009). Analyzing CUDA workloads using a detailed GPU simulator. *IEEE International Symposium On Performance Analysis Of Systems And Software,*, pp. 163--174.

Balkan, A. O., Horak, M. N., Qu, G. & Vishkin, U. (2007). Layout-accurate design and implementation of a high-throughput interconnection network for single-chip

parallel processing. *IEEE Symp. On High Performance Interconnection Networks, 2007*, pp. 21--28.

Barney, B. et al. (2010). Introduction to parallel computing. *Lawrence Livermore National Laboratory*, 6 (13), p. 10.

Barroso, L. A., Gharachorloo, K., Mcnamara, R., Nowatzyk, A., Qadeer, S., Sano, B., Smith, S., Stets, R. & Verghese, B. (2001). Piranha: a scalable architecture based on single-chip multiprocessing. *Proceedings Of The 27Th Annual International Symposium On Computer Architecture*, 28 (2), pp. 282-293. doi:10.1145/342001.339696.

Bekas, C. (2013). *Breaking the power wall in Exacale ComputingWhat is Exascale computing*. [e-book] Scientific colloquim, of Steinbuch Centre for Computing of Karlsruhe Institute of Technology f. Available through: http://www.scc.kit.edu http://www.scc.kit.edu/downloads/oko/SCC_Kolloquium_20131114_exascale_computing.pd [Accessed: 18 Apr 2013].

Birrer, S. (2008). *Addressing the limitations of tree-based approaches to high-bandwidth streaming multicast*. Proquest, Evanston Illinois.

Boden, N. J., Kulawik, A. E., Seitz, C. L., Cohen, D., Felderman, R. E., Seizovic, J. N. & Su, W. (1995). Myrinet: A gigabit-per-second local area network. *IEEE Micro*, 15 (1), pp. 29--36.

Bogdanski, B., Johnsen, B. D., Reinemo, S. & Flich, J. (2013). Making the network scalable: inter-subnet routing in infiniband. *Springer*, pp. 685--698.

Bouhraoua, A., Diraneyya, O. & Elrabaa, M. (2009). A simplified router architecture for the modified Fat Tree Network-on-Chip topology. *NORCHIP Proceedings*, Nov 2009 Trondheim: pp. 1--4.

Candaele, B., Aguirre, S., Sarlotte, M., Anagnostopoulos, I., Xydis, S., Bartzas, A., Bekiaris, D., Soudris, D., Lu, Z., Chen, X. et al (2010). Mapping Optimisation for Scalable multi-core ARchiTecture: The MOSART approach. *IEEE Annual Symposium On VLSI*, pp. 518--523.

Gropp (2014). Challenges for Algorithms and Software at Extreme Scale. [e-

presentation] invited talk at AICES, Aachen. Available through: http://www.cs.illinois.edu http://www.cs.illinois.edu/~wgropp/bib/talks/tdata/2013/siam-cse-invited.pdf.

Chen, D., Eisley, N., Heidelberger, P., Kumar, S., Mamidala, A., Petrini, F., Senger, R., Sugawara, Y., Walkup, R., Steinmacher-Burow, B. & Others (2012). Looking under the hood of the ibm blue gene/q network. *SC '12 Proceedings Of The International Conference On High Performance Computing, Networking, Storage And Analysis*.

Chiu, J., Chou, Y. & Chen, P. (2010). Hyperscalar: A novel dynamically reconfigurable multi-core architecture. *39Th International Conference On Parallel Processing*, pp. 277--286.

Chueh, H., Lien, C., Chang, C., Cheng, J. & Lee, D. (2013). Load-balanced Birkhoff-von Neumann switches and fat-tree networks. *IEEE 14Th International Conference On High Performance Switching And Routing (HPSR)*,Jul 2013 Taipei*:* pp. 142--147.

Cray Inc (2011). *Cray Supercomputers in Climate, Weather and Ocean Modeling*. [e-book] Cray.com. Available through: http://www.cray.com http://www.cray.com/Assets/PDF/products/xt/WP-XT02-1010.pdf [Accessed: 2 Jan 2013].

Cray Inc (2013). *Cray XK 7 Product brochure*. [e-brochure] Cray.com. Available through: www.cray.com http://www.cray.com/Products/Computing/XK7.aspx [Accessed: 2 Apr 2013].

Dally, W. J. & Towles, B. (2004). *Principles and practices of interconnection networks*. Amsterdam: Morgan Kaufmann Publishers.

Dally, W. J. & Towles, B. (2001). Route packets, not wires: On-chip interconnection networks. *Design Automation Conference*, pp. 684--689.

Das, R., Eachempati, S., Mishra, A. K., Narayanan, V. & Das, C. R. (2009). Design and evaluation of a hierarchical on-chip interconnect for next-generation CMPs. *HPCA*, pp. 175--186.

Davis, T. W., Liang, X., Navarro, M., Bhatnagar, D. & Liang, Y. (2012). An experimental study of WSN power efficiency: MICAz networks with Xmesh. *International Journal Of Distributed Sensor Networks*, 2012.

Dimakopoulos, V. V. & Hadjidoukas, P. E. (2011). HOMPI: A hybrid programming framework for expressing and deploying task-based parallelism. *Springer*, pp. 14--26.

Duato, J., Yalamanchili, S. & Ni, L. M. (2003). *Interconnection networks*. San Francisco, CA: Morgan Kaufmann.

El-Amawy, A. & Latifi, S. (1991). Properties and performance of folded hypercubes. *Parallel And Distributed Systems, IEEE Transactions On*, 2 (1), pp. 31--42.

Feng, C., Li, J., Lu, Z., Jantsch, A. & Zhang, M. (2011). Evaluation of deflection routing on various NoC topologies. *Proceedings Of 2011 9Th IEEE International Conference On ASIC.*, Oct 2011, Xiamen: pp. 163--166.

Feng, R., Zhang, P. & Deng, Y. (2012). Simulated Performance Evaluation of a 6D Mesh/iBT Interconnect. *13Th ACIS International Conference On Software Engineering, Artificial Intelligence, Networking And Parallel/Distributed Computing*, Aug 2012, Kyoto:pp. 253--259.

Garcia, M., Vallejo, E., Beivide, R., Odriozola, M., Camarero, C., Valero, M., Rodriguez, G., Labarta, J. & Minkenberg, C. (2012). On-the-fly adaptive routing in high-radix hierarchical networks. *41St International Conference On Parallel Processing*, Sep 2012 Pittsburg: pp. 279--288.

Gomez, C., Gilabert, F., Gomez, M. E., Lopez, P. & Duato, J. (2008). Beyond Fat--tree: Unidirectional Load--Balanced Multistage Interconnection Network. *IEEE Computer Architecture Letters*, 7 (2), pp. 49--52.

Grammatikakis, M. D., Hsu, D. F. & Kraetzl, M. (2001). *Parallel system interconnections and communications*. Boca Raton, FL: CRC Press.

Gratz, P. V. & Keckler, S. W. (2010). Realistic workload characterization and analysis for networks-on-chip design. *The 4Th Workshop On Chip Multiprocessor Memory Systems And Interconnects (CMP-MSI),*, pp. 1--10.

Gravenstreter, G. & Melhem, R. G. (1998). Realizing common communication patterns in partitioned optical passive stars (POPS) networks. *Computers, IEEE Transactions On*, 47 (9), pp. 998--1013.

Grun, P. (2010). Introduction to infiniband for end users. *White Paper, Infiniband Trade Association*.

Gupta, V., Gavrilovska, A., Schwan, K., Kharche, H., Tolia, N., Talwar, V. & Ranganathan, P. (2009). GViM: GPU-accelerated virtual machines. *Proceedings Of The 3Rd ACM Workshop On System-Level Virtualization For High Performance Computing*, pp. 17--24. doi:10.1145/1519138.1519141.

Hawkins, C., Small, B. A., Wills, D. S. & Bergman, K. (2007). The data vortex, an all optical path multicomputer interconnection network. *Parallel And Distributed Systems, IEEE Transactions On*, 18 (3), pp. 409--420.

Hennessy, J. L., Patterson, D. A. & Arpaci-Dusseau, A. C. (2012). *Computer architecture*. Watham: Elsevier/Morgan Kaufmann Publishers.

Hura, G. S. & Singhal, M. (2001). *Data and computer communications*. Boca Raton, FL: CRC Press.

IBM Bluewaters. (2014). *Blue Waters NCSA University of Illinois*. [online] Retrieved from: https://bluewaters.ncsa.illinois.edu/ [Accessed: 2 Apr 2013].

Kariniemi, H. & Nurmi, J. (2004). Performance evaluation and implementation of two adaptive routing algorithms for XGFT networks. *Computing And Informatics*, 23 (5-6), pp. 415--435.

Kariniemi, H. & Nurmi, J. (2006). On-Line Reconfigurable XGFT Network-on-Chip Designed for Improving the Fault-Tolerance and Manufacturability of the MPSoC Chips. *International Conference On Field Programmable Logic And Applications (FPL)*, Aug 2006 Madrid: pp. 1--6.

Kaur, T. & Aggarwal, R. (2012). *Adaptive Algorithms for Routing in Optical Interconnection Networks*. Saarbru¨cken: LAP LAMBERT Academic Publishing.

Khan, M., Ahmed, J. & Waqas, A. (2011). High performance scalable on-chip interconnects: Unleashing system performance. *Computer Networks And*

Information Technology (ICCNIT),, pp. 21--27.

Khosravi, A., Khors, I, S. & Akbari, M. K. (2011). Hyper node torus: A new interconnection network for high speed packet processors. *International Symposium On Computer Networks And Distributed Systems (CNDS*, pp. 106--110.

Kim, J., Dally, W. J. & Abts, D. (2007). Flattened butterfly: a cost-efficient topology for high-radix networks. *Proceedings Of The 34Th Annual International Symposium On Computer Science*, 35 (2), pp. 126--137.

Kim, J., Dally, W. J., Scott, S. & Abts, D. (2008). Technology-driven, highly-scalable dragonfly topology. *Proceedings Of The 35Th Annual International Symposium On Computer Architecture*, 36 (3), pp. 77--88.

Kini, G. N., Kumar, S. M. & Mruthyunjaya, H. (2009). Torus Embedded Hypercube Interconnection Network: A Comparative Study. *International Journal Of Computer Applications*, 1 (4), pp. 29--31.

Kinsy, M. A. (2009). Application-aware deadlock-free oblivious routing. *ACM SIGARCH Computer Architecture News*, pp 208-219

Koohi, S., Abdollahi, M. & Hessabi, S. (2011). All-optical wavelength-routed NoC based on a novel hierarchical topology. *Fifth IEEE/ACM International Symposium On Networks On Chip (Nocs)*, pp. 97--104.

Kumar, J. M. & Patnaik, L. M. (1992). Extended hypercube: A hierarchical interconnection network of hypercubes. *Parallel And Distributed Systems, IEEE Transactions*, 3 (1), pp. 45--57.

Leiserson, C. E. (1985). Fat-trees: universal networks for hardware-efficient supercomputing. *IEEE Transactions On Computers*, 100 (10), pp. 892--901.

Lin, X., Chung, Y. & Huang, T. (2004). A multiple LID routing scheme for fat-tree-based InfiniBand networks. *Proceedings Of 18Th International Parallel And Distributed Processing Symposium, 2004*, p. 11.

Liu, J., Ch, Rasekaran, B., Wu, J., Jiang, W., Kini, S., Yu, W., Buntinas, D., Wyckoff, P., P & A, D. K. (2003). Performance comparison of MPI implementations over

InfiniBand, Myrinet and Quadrics. *ACM/IEEE conference on supercomputing* Nov 2003: pp. 58.

Liu, Z., Zhang, X., Zhao, Y. & Guan, H. (2008). An asymptotically minimal node-degree topology for load-balanced architectures. *IEEE Global telecommunications conference.* Nov-Dec 2008 New Orleans:pp. 1--6.

Ludovici, D., Gilabert, F., Medardoni, S., Gomez, C., G'Omez, M. E., Lopez, P., Gaydadjiev, G. N. & Bertozzi, D. (2009). Assessing fat-tree topologies for regular network-on-chip design under nanoscale technology constraints. *Proceedings Of the Conference On Design, Automation And Test In Europe*, Apr 2009, Nice: pp. 562--565.

Martey, A. & Sturgess, S. (2002). *IS-IS network design solutions*. Indianapolis, IN: Cisco Press.

Mellanox Inc. (2012). Introduction to Cloud Design Four Design Principals For IaaS. *Mellanox Technologies, White Paper*, Retrieved from: http://www.mellanox.com/oem/dell/rel_docs/Cloud_Computing-1.pdf.

Mellanox Inc. (2003). Introduction to InfiniBand. *Mellanox Technologies, White Paper*, Retrieved from: http://www.cse.ohio-state.edu/~panda/5429/papers/2a_IB_Intro_WP_190.pdf.

Miller, J. E., Kasture, H., Kurian, G., Gruenwald, C., Beckmann, N., Celio, C., Eastep, J. & Agarwal, A. (2010). Graphite: A distributed parallel simulator for multicores. *High Performance Computer Architecture (HPCA), IEEE 16Th International Symposium*, pp. 1--12.

Minkenberg, C., Luijten, R. P. & Rodriguez, G. (2011). On the optimum switch radix in fat tree networks. *IEEE 12Th International Conference On High Performance Switching And Routing (HPSR)*, Jul 2011, Cartagena: pp. 44--51.

Montanana, J. M., Koibuchi, M., Matsutani, H. & Amano, H. (2009). Balanced Dimension-Order Routing for k-ary n-cubes. *International Conference On Parallel Processing Workshop (ICPPW)*.Sep. 2009, IEEE: pp. 499--506.

Mu, Y. & Li, K. (2011). Extended Folded Cube: A Improved Hierarchical

Interconnection Network. *Fourth International Symposium On Parallel Architectures*, pp. 77--81.

Pal, R. K., Paul, K. & Prasad, S. (2012). ReKonf: A Reconfigurable Adaptive ManyCore Architecture. *IEEE 10Th International Symposium On Parallel And Distributed Processing With Applications (ISPA)*, pp. 182--191. doi:10.1109/ISPA.2012.32.

Pasricha, S. & Dutt, N. (2008). *On-chip communication architectures*. Amsterdam: Elsevier / Morgan Kaufmann Publishers.

Peratikou, A. & Adda, M. (2014). OPTIMISING EXTENDED GENERALISED FAT TREE TOPOLOGIES. *Distributed Computer And Communication Networks 17Th International Conference*, Oct 2013 Moscow: pp. 82-90.

Peratikou, A. & Adda, M. (2014). Optimisation of Extended Generalised Fat Tree Topologies. *Springer*, pp. 82--90.

Pericas, M., Cristal, A., Carzola, F., Gonzales, R., Gimenez, D. & Valero, M. (2007). "A Flexible Heterogeneous Multi-Core Architecture", paper presented at *International conference Parallel Architecture and Compilation Techniques (PACT)*. Barcelona Supercomputing Center, 15-19 Sept. 2007. Brasov: pp. 13-24.

Petrini, F. & Vanneschi, M. (1998). Performance analysis of wormhole routed k-ary n-trees. *International Journal Of Foundations Of Computer Science*, 9 (02), pp. 157--177.

Pfister, G. F. (2001). An introduction to the InfiniBand architecture. *High Performance Mass Storage And Parallel I/O*, 42 pp. 617--632.

Rahman, M. H. & Horiguchi, S. (2003). HTN: a new hierarchical interconnection network for massively parallel computers. *IEICE TRANSACTIONS On Information And Systems*, 86 (9), pp. 1479--1486.

Rahman, M. H., Jiang, X., Masud, M. & Horiguchi, S. (2009). Network performance of pruned hierarchical torus network. *Sixth IFIP International Conference On Network And Parallel Computing*, Oct 2009.Gold Coast: pp. 9--15.

Rahman, M., Inoguchi, Y., Faisal, F. A. & Kundu, M. K. (2011). Symmetric and Folded

Tori Connected Torus Network. *Journal Of Networks*, 6 (1).

Rahmani, A., Afzali-Kusha, A. & Pedram, M. (2009). A novel synthetic traffic pattern for power/performance analysis of network-on-chips using negative exponential distribution. *Journal Of Low Power Electronics*, 5 (3), pp. 396--405.

Reed, C. (1999). *Multiple Level Minimum Logic Network*, USA Patent US Patent 5 996 020,. [Accessed: 2 Apr 2012].

Roca, A., Hern', Ez, C., Flich, J., Silla, F. & Duato, J. (2011). A distributed switch architecture for on-chip networks. *International Conference On Parallel Processing (ICPP)*, Sep 2011.Tapei: pp. 21--30.

Shafiee, A., Zolghadr, M., Arjom & Sarbazi-Azad, H. (2011). Application-aware deadlock-free oblivious routing based on extended turn-model. *IEEE International conference on Computer aided design (ICCAD).* Nov 2011.San jose: pp. 213--218.

Shi, L., Chen, H., Sun, J. & Li, K. (2012). vCUDA: GPU-accelerated high-performance computing in virtual machines. *IEEE Transactions On Computers*, 61 (6), pp. 804--816. doi:10.1109/TC.2011.112.

Shi, L., Chen, H., Sun, J. & Li, K. (2009). vCUDA: GPU-accelerated high-performance computing in virtual machines. *IEEE International Symposium On Parallel And Distributed Processing*, pp. 1-11. doi:10.1109/IPDPS.2009.5161020.

Sivaram, R., Govindaraju, R. K., Hochschild, P., Blackmore, R. & Chaudhary, P. (2005). Breaking the connection: Rdma deconstructed. *Hot Interconnects 2005*, pp. 36--42.

Sivaram, R., Stunkel, C. B., P & A, D. K. (2002). HIPIQS: A high-performance switch architecture using input queuing. *Parallel And Distributed Systems, IEEE Transactions On*, 13 (3), pp. 275--289.

Skiee (2012). *Fat trees and dragon fly: a perspective on topologies*. [e-book] Simula Research Laboratory, HPC: Available through: http://www.hpcadvisorycouncil.com Retrieved from : http://www.hpcadvisorycouncil.com/events/2012/European-Workshop/Presentations/5_Simula.pdf March 2013.

Task, C. & Chauhan, A. (2008). A model for communication in clusters of multi-core machines. *4Th IEEE International Conference On Escience*, pp. 356--357.

Venkateswaran, N., Saravanan, K. P., Nachiappan, N. C., Vasudevan, A., Subramaniam, B. & Mukundarajan, R. (2010). Custom built heterogeneous multi-core architectures (cubemach): Breaking the conventions. *IEEE International Symposium On Parallel & Distributed Processing, Workshops And Phd Forum (IPDPSW)*, pp. 1--15.

WANG, L., XU, Z., CHEN, Y. & WANG, X. (2009). Research and Application of XMesh Networking Technology [J]. *Journal Of Changshu Institute Of Technology*, 10 p. 024.

Xiao-Jing, Z., Wei-Wu, H. & Ke, M. (2014). Xmesh: A Mesh-Like Topology for Network on Chip. *Journal Of Software*, 18 pp. pp. 2194-2204. doi:10.1360/jos182194.

Xie, M., Lu, Y., Wang, K., Liu, L., Cao, H. & Yang, X. (2012). Tianhe-1a interconnect and message-passing services. *IEEE Micro*, 32 (1), pp. 08--20.

Yang, Q. (2009). Performance evaluation of k-ary data vortex networks with bufferless and buffered routing nodes. *, Proc. SPIE 7633, Network Architectures, Management, And Applications VII, 76331K*, pp. 76331--76331. doi:10.1117/12.851813.

Yang, X. & Lee, R. B. (2000). Fast subword permutation instructions using omega and flip network stages. *Proceedings Of 2000 International Conference In Computer Design.* Sep 2000, Austin Texas: pp. 15--22.

Yang, X., Liao, X., Lu, K., Hu, Q., Song, J. & Su, J. (2011). The TianHe-1A supercomputer: its hardware and software. *Journal Of Computer Science And Technology*, 26 (3), pp. 344--351.

Youssef, A. & Narahari, B. (1992). Topological properties of generalized banyan-hypercube networks. *Journal Of Parallel And Distributed Computing*, 14 (1), pp. 98--103.

Youyao, L., Jungang, H. & Huimin, D. (2008). A Hypercube-based Scalable

Interconnection Network for Massively Parallel Computing. *Journal Of Computers*, 3 (10), pp. 58-65.

Yu-Hang, L., Ming-Fa, Z., Jue, W., Li-Min, X. & Tao, G. (2012). Xtorus: An Extended Torus Topology for On-Chip Massive Data Communication. *IEEE 26Th International Parallel And Distributed Processing Symposium Workshops & Phd Forum*, pp. 2061--2068.

Zhang, C. & Li, M. (2011). P2i-torus: A hybrid architecture for direct interconnection. *International Conference On Computer Science And Network Technology*, 3 pp. 2042--2046.

Zhang, P., Powell, R. & Deng, Y. (2011). Interlacing bypass rings to torus networks for more efficient networks. *Parallel And Distributed Systems, IEEE Transactions On*, 22 (2), Dec 2011, Harbin: pp. 287--295.

Ziakas, D., Baum, A., Maddox, R. A. & Safranek, R. J. (2010). Inteltextregistered quickpath interconnect architectural features supporting scalable system architectures. *18Th IEEE Symposium On High Performance Interconnects*, pp. 1--6.

Appendix A-Latency equations proof

Latency analysis of fat tree class routing addressing based in Virtual cut through and wormhole routing .

7.4. A.1 Latency

A message that reaches level i of the tree moves sides to another Znode or downwards.

The full message length will include the routing, the address overhead along with the payload.

Δ is the propagation delay between any two switches at level i, D is the data payload of the packet, (ρ) is the routing information bit to find the common ancestor, a is the overhead address to determine the Znode .(d) is defined as the extra propagation delay that is incurred when the message is forwarded to the next Znode. Finally (bi) is the address overhead used to determine the next zone.

For evenly distributed traffic:

$$\mathrm{T_{m=1}} = \frac{1}{\mathrm{n}} \sum_{l=1}^{n} T_i$$

Equation 32

For level 1 it would be: $T_1 = \Delta + \rho + b_1 + \mathrm{D}$

For level 2: $T_2 = 3\Delta + 2\rho + b_1 + b_2 + \mathrm{D}$

For level 3: $T_3 = 5\Delta + 3\rho + b_1 + b_2 + b_3 + \mathrm{D}$

For level n: $T_n = (2n) * \Delta + n\rho + b_n + b_{n-1} + b_{n-2} + \cdots + b_2 + b_1 + \mathrm{D}$

So the general latency would be

$$T_{m=1} = \Delta * (n+1) + \frac{1}{2} * (n+1) * \rho + D + \frac{1}{n} * \sum_{j=1}^{n} (n-j+i) b_j$$

Where for sliced addressing the ρ would be equal to 1 and $b_j = \log_2 Z_n$

When more than one Zoned nodes exist then the latency would be :

$$T_{m>1} = \frac{1}{2}\left(\frac{1}{n}\sum_{l=1}^{n} T_l^I + \frac{1}{n}\sum_{l=1}^{n} T_l^E = \frac{1}{2} * (T_l^I + T_l^E)\right)$$

Which can be fully expressed for sliced addressing as :

$$T_{n\_S} = \Delta * (n+1) + \frac{1}{2} * (n+1) * \rho + D + \frac{1}{n} * \sum_{j=1}^{n} (n-j+i) b_j$$

And similarly for destination addressing we would have the same equation but with replacing the routing bit with $log_2 P$.

$$T_{n\_D} = \Delta * (n+1) + \frac{1}{2} * (n+1) * log_2 P + \frac{1}{n} * \sum_{j=1}^{n-1} (n-j+i) b_j + D$$

For source destination the same equation applies as destination without the devision in the number of levels as the source destination has slightly higher latency than destination only addressing

$$T_{n\_SD} = \Delta * (n+1) + (n+1)\, log_2 P + \frac{1}{n} * \sum_{j=1}^{n-1} (n-j+i) b_j + D$$

Where b_j can either be 0 or greater than 1 depending on the addressing mode used, For instance on source destination addressing routine, the Bj does not exist to all the configurations for instance if the traffic is going to the same node the bj is not needed therefore bj=0 but when the traffic is going to another m node then the bi is needed as the new node will not have the addressing that is required to complete the transmission downwards. Refer to table 3.1 in chapter 3.

7.5. A.2 Flat addressing latency:

When only one Znode exist the flat addressing can be expressed with the equation 1.

Since in flat addressing everything reaches the highest level which is level n we can generalized it as:

$$T_{m=1} = (2 * n) * \Delta + log_2 P + D$$

Where (Δ) propagation is for the upward direction that the packet will route, $(\Delta + \log_2 p)$ is the final decision that once the top level is reached the packet will be required to move downwards to complete the transmission.

When the Zoned nodes are more than one then we have to include some extra parameters.

$$T_{m>1} = \frac{1}{2}\left(\frac{1}{n}\sum_{l=1}^{n} T_i^I + \frac{1}{n}\sum_{l=1}^{n} T_i^E = \frac{1}{2} * (T_i^I + T_i^E)\right)$$

Where T_i^I is the internal latency and T_i^E the external to identify wether the message is forwarded to a different Zoned node or stays to the same one. If it stays to the same one then T_i^E would be equal to 0. The equation can be written in a more general form as:

$$T_{m>1} = T_n + \frac{\Delta}{2} + log_2 m$$

Where the general latency is $T_n = \frac{1}{n}\sum_{i=1}^{n} T_i$ as illustrated on equation 1 and $\log_2 m$ is to identify the Zoned nodes.

Weight of latencies:

For level n the actual weight of is W1

$$T = \frac{W_1T_1+W_2T_2+W_3T_3+\cdots+W_nT_n}{W_1+W_2+W_3+\cdots+W_n}$$

Therefore: $T = \sum_{i=1}^{n} W_i * T_i$

When the traffic is evenly distributed across all levels then for slice and Source destination and destination addressing only:

$$W_i = \frac{1}{n} \ , \ log_2 Z_i = 0$$

For flat we can have: $W_1 = W_2 \ldots = W_{n-1} = 0$

Where $W_n = 1$

Appendix B-Upwards & downwards traffic

This appendix focuses in exposing the proofs for the traffic formulas.

The function of connectivity plays an important part in the traffic. Two traffic modes exit, the up traffic and the down traffic.

B.1 Down traffic:

The traffic that reaches the top switch with probability q_i is the routing switch R_n, and since $n = \frac{Z_i Z_2 \dots Z_n, q_1, q_2, \dots (1-q_n)\lambda}{R_n}$, where $q_n = 0$.

The traffic entering each R_{n-1} at level $n-1$ is $\frac{Z_i Z_2 \dots Z_n \lambda}{Z_n R_{n-1}} = \frac{Z_i Z_2 \dots Z_{n-1}}{R_{n-1}}$

At level i the traffic between the routing switches R_{n-1} from the top switching element is $\frac{Z_i Z_2 \dots Z_i \lambda}{R_i}$ which must be less than down capacity links of the routing switches R_i.

Therefore: $\frac{Z_i Z_2 \dots Z_i}{R_i} \leq \begin{cases} Z_i \ \psi_i^+ \mu & positive \\ \frac{R_{i-1}}{R_i} Z_i \psi_i^- \mu & negative \\ R_{i-1} Z_i \mu & full \end{cases}$

For level 1:

$$\frac{Z_1}{R_1} \leq \begin{cases} Z_1 \ \psi_1^+ & positive \\ \frac{R_0}{R_1} Z_1 \psi_1^- & negative \\ R_0, Z_1 & full \end{cases}$$

R_0 is a virtual variable and can be used for convenience as : $R_0 = R_1$ for negative connectivity and $R_0 = 1$ for full connectivity and the connectivity degree should be $\psi_1^- = \psi_1^+ = 1$.

Therefore $R_i \geq 1$ derives as a constrain.

For level i: The above equations can be generalized for omega connectivity as

$$Z_i \leq \frac{\begin{Bmatrix} R_{i+1} & \psi_{i+1}^{+} \\ R_i & \psi_{i+1}^{-} \\ R_i, R_{i+1} & \end{Bmatrix}}{\prod_{j=1}^{i-1} Z_j}$$

Which can be simplified for each connectivity separately for positive connectivity as:

$Z_i \leq \frac{R_{i+1} \times \psi_{i+1}}{\prod_{j=i}^{n} Z_j}$ For negative connectivity as $Z_i \leq \frac{R_i \times \psi_{i+1}}{\prod_{j=i}^{n} Z_j}$.

B.2 Up traffic:

For level 1:

$$\frac{Z_1}{R_1}\lambda q_1 \leq \begin{cases} \mu\left(\frac{R_2}{R_1}\right) \ \psi_2^{+} & positive \\ \mu\, \psi_2^{-} & negative \\ \mu\, R_2 & full \end{cases}$$

For level i

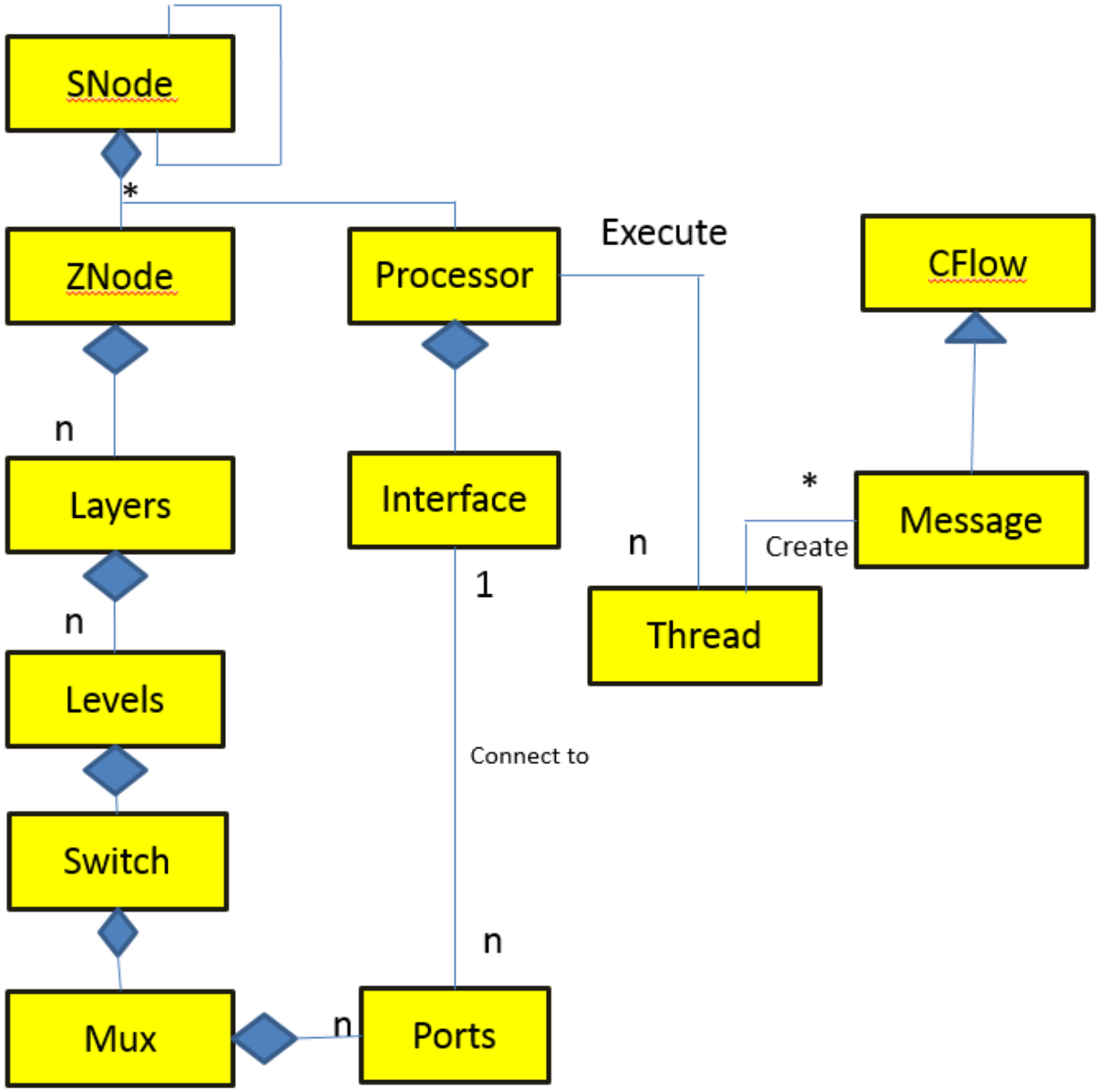
$$\frac{Z_1 Z_2 \ldots Z_\iota\, q_1 q_2 \ldots q_i\, \lambda}{R_i} \leq \begin{cases} \mu\left(\frac{R_{i+1}}{R_i}\right) \ \psi_{i+1}^{+} & positive \\ \mu\, \psi_{i+1}^{-} & negative \\ \mu\, R_{i+1} & full \end{cases}$$

Thus for $1 \leq i \leq n$ the top switch has no Upward traffic therefore it would be as follow which is the same as the down traffic.

$$Z_i \leq \frac{\begin{Bmatrix} R_{i+1} & \psi_{i+1}^{+} \\ R_i & \psi_{i+1}^{-} \\ R_i, R_{i+1} & \end{Bmatrix}}{\prod_{j=1}^{i-1} Z_j}$$

Appendix C- Galanet class diagram

The class diagram of the Galanet simulator is illustrated in the following figure.



Appendix D- Simulator parameters

This appendix focuses in illustrating the parameters text file for the Galanet simulator.

```
***GALAXY_NETWORK_CONFIGURATION_DATA***
NUMBER_OF_TIERS              = 1
NUMBER_OF_SUPER_NODES_PER_TIER = 1
NUMBER_OF_ZONED_NODES        = 1
NUMBER_OF_LEVELS_PER_ZNODE    = 3
NUMBER_OF_LAYERS_PER_ZNODE    = 1
NUMBER_OF_SWITCHES_PER_LEVEL_PER_ZONE             = 1 8 64
NUMBER_OF_ZONES_UNDER_EACH_LEVEL                  = 8 8 8
CONNECTIVITY_DEGREE_PER_SWITCH_PER_LEVEL          = 1 1 1
CONNECTIVITY_FUNCTION_PER_LEVEL(Butterfly=0,Omega=1) = 0 0 0
CONNECTIVITY_BETWEEN_LAYERS_OF_ZNODE(no=0,both=1,down=2,up=3) =
0 0 0
CONNECTIVITY_BETWEEN_LAYERS_OF_SNODE(no=0,yes=1)           = 0 0 0
CHANNELS_PER_UP_PORT              = 1 1 1
CHANNELS_PER_SIDE_PORT            = 1 1 1
CHANNELS_PER_DOWN_PORT            = 1 1 1
TRAFFIC_PERCENTAGE_AT_EACH_LEVEL    = 0 0 0
NUMBER_OF_THREADS                 = 1
PAYLOAD_LENGTH(Bits)              = 32
INTER_ARRIVAL_TIME(nano-seconds)   = 64
STARTING_TIME(nano-seconds)       = 20
PAYLOAD_DISTRIBUTION(Constant=1,Uniform=2,Exponential=3)                = 2
INTER_ARRIVAL_TIME_DISTRIBUTION(Constant=1,Uniform=2,Expo1ential=3)
= 3
STARTING_TIME_DISTRIBUTION(Constant=1,Uniform=2,Exponential=3)          =
1
DESTINATION(Random=1,RoundRobin=2,Complement=3,BitReverssal=4,Transpose
=5,HotSpotDestination=6) = 5
PERCENTAGE_HOT_SPOT_TRAFFIC = 640
ADDRESSING_MODE(SlicedFlits=0,Flat=1,SRCDST=2,DST=3)              = 3
ROUTING(Adaptive=0,Deterministic=1,Semi=2,Hybrid=3,Unidirectional=4) = 0
PERCENTAGE_HYBRID_ADAPTIVE_TO_DETERMINISTIC                       = 50
SWITCH_OPERATION(VT=0,SF=1)                                 = 0
BUFFER_SIZE(Wormwhole=0)                                    = 2
PORT_SCHEDULING(AllRoundRobin=0,GroupRoundRobin=1)                = 1
CHANNEL_RATE(Gbits/sec) = 1
PROPAGATION_DELAY(Bits) = 1
PRINT_DETAILS_STATISTICS(No=0,Yes=1) = 0
PRINT_NETWORK_MAP(No=0,Yes=1)       = 0
RECORD_ANIMATION(No=0,Yes=1)        = 0
```

Appendix E- Network map and statistics

In this appendix we illustrate the network map file followed by the statistics file for an example system with 8 processors and 6 switches.

E.1 Network map file 8 processors 6 switches

Node 0 :

Level :1

Switch [0][1][0] (0102B3EC) Connect to Processor 0 [0102A4BC], Processor 1 [0102A56C], Processor 2 [0102A61C], Processor 3 [0102A6CC],

Switch [0][1][1] (0102B410) Connect to Processor 4 [0102A77C], Processor 5 [0102A82C], Processor 6 [0102A8DC], Processor 7 [0102A98C],

Level :2

Switch [0][2][0] (0102B2DC) Connect down to

Switch [0][1][0] (0102B3EC) Connects up to Switch [0][2][0] (0102B2DC), Switch [0][2][1] (0102B300), Switch [0][2][2] (0102B324), Switch [0][2][3] (0102B348),

Switch [0][1][1] (0102B410) Connects up to Switch [0][2][0] (0102B2DC), Switch [0][2][1] (0102B300), Switch [0][2][2] (0102B324), Switch [0][2][3] (0102B348),

Switch [0][2][1] (0102B300) Connect down to

Switch [0][1][0] (0102B3EC) Connects up to Switch [0][2][0] (0102B2DC), Switch [0][2][1] (0102B300), Switch [0][2][2] (0102B324), Switch [0][2][3] (0102B348),

Switch [0][1][1] (0102B410) Connects up to Switch [0][2][0] (0102B2DC), Switch [0][2][1] (0102B300), Switch [0][2][2] (0102B324), Switch [0][2][3] (0102B348),

Switch [0][2][2] (0102B324) Connect down to

Switch [0][1][0] (0102B3EC) Connects up to Switch [0][2][0] (0102B2DC), Switch [0][2][1] (0102B300), Switch [0][2][2] (0102B324), Switch [0][2][3] (0102B348),

Switch [0][1][1] (0102B410) Connects up to Switch [0][2][0] (0102B2DC), Switch [0][2][1] (0102B300), Switch [0][2][2] (0102B324), Switch [0][2][3] (0102B348),

Switch [0][2][3] (0102B348) Connect down to

Switch [0][1][0] (0102B3EC) Connects up to Switch [0][2][0] (0102B2DC), Switch [0][2][1] (0102B300), Switch [0][2][2] (0102B324), Switch [0][2][3] (0102B348),

Switch [0][1][1] (0102B410) Connects up to Switch [0][2][0] (0102B2DC), Switch [0][2][1] (0102B300), Switch [0][2][2] (0102B324), Switch [0][2][3] (0102B348),

E.2 Statistics file for 8 processors 6 switches

----------Detailed Statistics----------

Zb Node : 0

Processor : 0

Average queue = 0.833333 Average waiting time = 48.0551 Maximum waiting time = 65.95

Interface [0]:

Input Rate = 0.497846 Sent messages = 13 Received messages = 16

Processor : 1

Average queue = 0.75 Average waiting time = 32.8174 Maximum waiting time = 80.746

Interface [0]:

Input Rate = 0.461371 Sent messages = 10 Received messages = 8

Processor : 2

Average queue = 0.75 Average waiting time = 44.0791 Maximum waiting time = 62.8203

Interface [0]:

Input Rate = 0.370674 Sent messages = 12 Received messages = 10

Processor : 3

Average queue = 0.75 Average waiting time = 29.9317 Maximum waiting time = 65.2511

Interface [0]:

Input Rate = 0.334198 Sent messages = 13 Received messages = 8

Processor : 4

Average queue = 0.75 Average waiting time = 23.982 Maximum waiting time = 35.2598

Interface [0]:

Input Rate = 0.239558 Sent messages = 7 Received messages = 13

Processor : 5

Average queue = 0.833333 Average waiting time = 66.5522 Maximum waiting time = 107.387

Interface [0]:

Input Rate = 0.476158 Sent messages = 13 Received messages = 10

Processor : 6

Average queue = 0.5 Average waiting time = 13.4138 Maximum waiting time = 33.417

Interface [0]:

Input Rate = 0.473201 Sent messages = 16 Received messages = 15

Processor : 7

Average queue = 0.5 Average waiting time = 28.3539 Maximum waiting time = 28.3539

Interface [0]:

Input Rate = 0.222799 Sent messages = 8 Received messages = 12

Zb Node : 0

Layer 0:

Level : 1

Switch 0:

Upper central Queue: 0 Average queue = 0 Average waiting time = 0 Maximum waiting time = 0 Maximum waiting in Que = 0

Upper Port 0 : Utilisation = 0.26026

Upper Port 1 : Utilisation = 0.240544

Upper Port 2 : Utilisation = 0.193224

Upper Port 3 : Utilisation = 0.21294

lower Central Queue 0: Average queue = 1.16667 Average waiting time = 40.5287 Maximum waiting time = 83.3039 Maximum waiting in Que = 3

Lower Port 0 : Utilisation = 0.504747

lower Central Queue 1: Average queue = 0.833333 Average waiting time = 33.8621 Maximum waiting time = 49.4875 Maximum waiting in Que = 2

Lower Port 0 : Utilisation = 0.268147

lower Central Queue 2: Average queue = 0.5 Average waiting time = 7.69081 Maximum waiting time = 8.1194 Maximum waiting in Que = 1

Lower Port 0 : Utilisation = 0.21294

lower Central Queue 3: Average queue = 0.5 Average waiting time = 8.85555 Maximum waiting time = 8.85555 Maximum waiting in Que = 1

Lower Port 0 : Utilisation = 0.315467

Switch 1:

Upper central Queue: 0 Average queue = 0 Average waiting time = 0 Maximum waiting time = 0 Maximum waiting in Que = 0

Upper Port 0 : Utilisation = 0.18928

Upper Port 1 : Utilisation = 0.197167

Upper Port 2 : Utilisation = 0.0985834

Upper Port 3 : Utilisation = 0.232657

lower Central Queue 0: Average queue = 0.666667 Average waiting time = 28.5142 Maximum waiting time = 59 Maximum waiting in Que = 2

Lower Port 0 : Utilisation = 0.33124

lower Central Queue 1: Average queue = 0.5 Average waiting time = 24.8854 Maximum waiting time = 24.8854 Maximum waiting in Que = 1

Lower Port 0 : Utilisation = 0.181394

lower Central Queue 2: Average queue = 0.7 Average waiting time = 20.593 Maximum waiting time = 37.8083 Maximum waiting in Que = 2

Lower Port 0 : Utilisation = 0.488974

lower Central Queue 3: Average queue = 1.07143 Average waiting time = 46.7897 Maximum waiting time = 87.3885 Maximum waiting in Que = 3

Lower Port 0 : Utilisation = 0.40222

Level : 2

Switch 0:

lower Central Queue 0: Average queue = 0 Average waiting time = 0 Maximum waiting time = 0 Maximum waiting in Que = 0

Lower Port 0 : Utilisation = 0.17745

lower Central Queue 1: Average queue = 0 Average waiting time = 0 Maximum waiting time = 0 Maximum waiting in Que = 0

Lower Port 0 : Utilisation = 0.244487

Switch 1:

lower Central Queue 0: Average queue = 0 Average waiting time = 0 Maximum waiting time = 0 Maximum waiting in Que = 0

Lower Port 0 : Utilisation = 0.185337

lower Central Queue 1: Average queue = 0 Average waiting time = 0 Maximum waiting time = 0 Maximum waiting in Que = 0

Lower Port 0 : Utilisation = 0.226742

Switch 2:

lower Central Queue 0: Average queue = 0 Average waiting time = 0 Maximum waiting time = 0 Maximum waiting in Que = 0

Lower Port 0 : Utilisation = 0.0887251

lower Central Queue 1: Average queue = 0 Average waiting time = 0 Maximum waiting time = 0 Maximum waiting in Que = 0

Lower Port 0 : Utilisation = 0.179422

Switch 3:

lower Central Queue 0: Average queue = 0 Average waiting time = 0 Maximum waiting time = 0 Maximum waiting in Que = 0

Lower Port 0 : Utilisation = 0.222799

lower Central Queue 1: Average queue = 0 Average waiting time = 0 Maximum waiting time = 0 Maximum waiting in Que = 0

Lower Port 0 : Utilisation = 0.20111

Appendix F Ethical review form